UNIVERSITY OF
EASTERN FINLAND

University of Eastern Finland

School of Computing

Master's Thesis

Detecting user actions in MOPSI

Radu Mariescu-Istodor

26th of September, 2013

ABSTRACT

Computing devices are becoming highly mobile. A study made by ABI Research[1] predicts that by the end of 2013 approximately 1.4 billion smartphones will be in use. Most devices have built-in GPS sensors which can estimate user's geographical position with uncertainty of a couple of meters.

User actions describe a user's behavior at a certain time. Location-based actions take the user location into account. For example:

1. *Andrei* is in *Joensuu, Finland*;
2. *Pasi* and *Mohammad* met at *Science Park.*

MOPSI[2] is a software application that helps users to find where their friends are and what is around. It allows photo sharing, easy tracking, and chatting between friends. The user actions described in this thesis are tailored for the MOPSI application as part of its *action detection module*. Example actions include login (logout), recording routes, publishing photos and meetings between users. Problems such as loss of network connectivity and poor location accuracy are also identified and dealt with.

MOPSI users are notified in real-time when a user action is detected. *Publish-subscribe* is a software architecture where the sender of a message, called *publisher* does not specifically identify a receiver. Instead, the messages are categorized into classes for which subscribers may express their interest. A particular user may find only a subset of all detected actions relevant. MOPSI users can therefore *subscribe* to get *notifications* when a particular type of action happens. For example, a user can subscribe to receive notifications when somebody takes pictures and records routes. These subscriptions are handled by the *notification module*.

The action detection and notification modules provide MOPSI users a way to keep up to date with what their friends are doing. The goal is to provide useful information in an easy way.

Keywords: user action, notification, agglomerative clustering, publish-subscribe, mobile user, location-tagged services

[1] http://www.businessinsider.com/15-billion-smartphones-in-the-world-22013-2
[2] http://cs.uef.fi/mopsi

ACKNOWLEDGEMENTS

I am grateful to the University of Eastern Finland and to all the teachers who helped me to gain experience in different areas of computer science. I was given the chance to be part of the IMPIT program and meet many students from different cultures, however, having similar background. Participating in countless activities and projects meant many unforgettable experiences. I want to express my thanks to all the organizers of IMPIT, 2011 for making all of it possible.

I wish to thank my supervisor, Professor Pasi Fränti, for his guidance together with the literally hundreds of observations and advices to both my research and work. I believe my time management, problem solving and presentation skills improved significantly thanks to his efforts. In addition I would like to thank every member of the MOPSI group for making me feel as part of a team.

I want to express my deepest gratitude to my girlfriend, family and friends for their moral support and encouragement. They are always by my side, supporting me, no matter the circumstances.

I would also like to thank my high school computer science teacher, Ionel Pițˇ-Rada, for showing me a possible career to pursue in life, and for always making time when I unexpectedly drop by during his work hours to discuss life choices and philosophy.

In the end, I would like to dedicate this thesis to my aunt, who regretfully passed away earlier this year. She will always be alive in my heart.

TABLE OF CONTENTS

1 Introduction

Computing devices have become ubiquitous and increasingly mobile in the past decade as we can see in studies such as the one made by ABI Research[3], a leading market intelligence company. Another study by ABI Research[4] shows that 85% of the devices ship with GPS sensors. Therefore, most devices can have access to their geographical position consisting of the latitude and longitude.

Car navigation systems use road data and suggest a path to a specific destination. Modern digital cameras such as Canon EOS 6D can store location information as meta-data in the image file. This data is standard and can be interpreted by other applications where it can be useful. For example digital photo albums can group photos according to their location or place them on exact location in a digital map.

Smartphones have different ways of obtaining the location information:

- GPS;
- Network;
- Wi-Fi.

***G**lobal **P**ositioning **S**ystem* (*GPS*) is a navigation system that relies on satellites to obtain location and time information. The satellites transmit timing signals and position data. A GPS receiver decodes these signals and interprets the arrival times in terms of latitude, longitude and altitude with an uncertainty which may be as small as a couple of meters. A minimum of four satellites is required for the receiver to handle the calculations using trilateration[5].

The *Network* and *Wi-Fi* systems work when the device is sensed by the network. The location is calculated based on the strength of the signal and the round-trip time to each tower. Accuracy is much lower than when using GPS and no altitude information is available. Using cell tower triangulation it is possible to determine a phone location to within an area of about ¾ of a square kilometer[6]. Better accuracy is achieved in densely populated urban areas where the cell towers are closer together.

[3] http://www.businessinsider.com/15-billion-smartphones-in-the-world-22013-2
[4] http://www.microwave-eetimes.com/en/majority-of-smartphones-to-feature-gps-accelerometers-and-gyroscopes-by-2013.html?cmp_id=7&news_id=222901138
[5] http://electronics.howstuffworks.com/gadgets/travel/gps.htm
[6] http://searchengineland.com/cell-phone-triangulation-accuracy-is-all-over-the-map-14790

Because *GPS* is not available indoors, the other two methods still have a purpose. Smartphones with positioning capabilities usually use all the three methods described above.

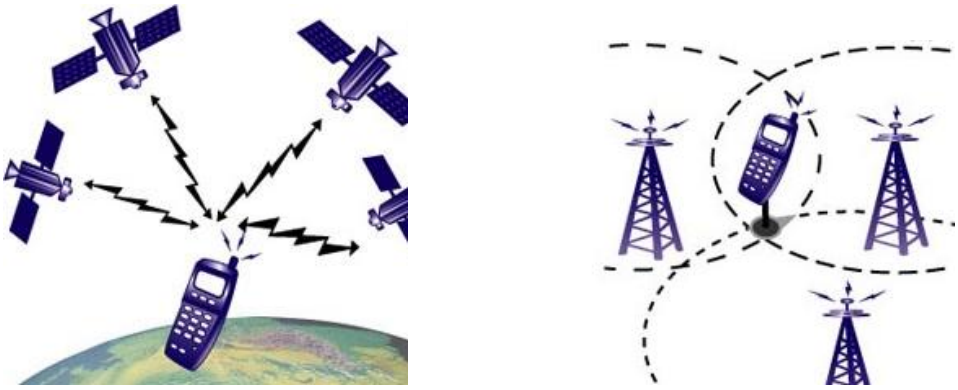


Figure 1: GPS positioning vs. Network / Wi-Fi positioning [7]

User actions are statements describing a user's behavior at a certain time. Detecting location-based actions is important in various situations. It can *help social interaction* by keeping users aware of how, when and where others interact. For example, social networks such as Facebook[8] allow users to share their current location or explicitly name the place they are visiting and the people they are meeting with. We aim to detect these actions automatically and notify other users about their presence.

Figure 2: An example of user actions on Facebook

[7] http://www.e-cartouche.ch/content_reg/cartouche/LBStech/en/html/LBStechU2_poslabel1.html
[8] http://www.facebook.com

Another example is *optimizing military configurations* by detecting unexpected situations automatically. Any unplanned action can be notified to the supervisor who can take immediate action and give new orders. This example is nicely illustrated in strategy games such as *Warcraft* where the player is notified of different actions such as attacking and retreating of the enemy army.

Observation of animal migrations and interactions is important to wildlife researchers. Animals are at constant threat due to fencing, highways, housing development, agriculture, energy development, wind turbines and many other manmade factors. Because of this it is important to keep track of their whereabouts and identify movement patterns which pose a potential risk and take action. Herds can be monitored by tracking a small sample of the population.

Figure 3: Herd of Buffalo[9]

Digital tour guides are a used at different tourist attractions. They help tourists organize themselves and manage tours without the help of a human guide. Digital guides can inspect the user actions to detect his or her interests and suggest a path for the rest of the visit or warn when a must see artifact is close and not yet visited. As an example we take a visitor who appears to be interested in art of the renaissance and is leaving the Louvre museum. If he or she has not seen the *Mona Lisa*, a gentle reminder could be issued and directions to the painting can be pointed out.

[9] http://wallpaperweb.org/wallpaper/animals/aerial-view-of-a-herd-of-african-buffalo-botswana_38390.htm

Notifying actions in real-time is needed when the information needs to reach others at the moment it happens. The military and wildlife examples mentioned above need the notifications system in order to be useful. *Publish-subscribe* is a software architecture widely used when senders of messages, called *publishers* do not specifically target a receiver. Instead, the messages are categorized into classes for which *subscribers* may express interest. A particular user may find only a subset of all detected actions relevant and will therefore subscribe to that subset. The only drawback of publish-subscribe architecture [1] is that users performing an action can expect the information to reach a certain user which may not be subscribed to that particular action type.

MOPSI[10] is a software application that helps users to find where their friends are and what is around. It allows photo sharing, easy tracking, and chatting between friends. This thesis documents the *action detection* and *notification* modules of the MOPSI system.

Many researchers have studied the detection and notification of user actions using mobile devices. In [2] and [3] the authors describe how publish-subscribe architecture can be extended to support mobile and location-dependent applications. The authors introduce *location-dependent subscriptions* as a way to filter actions keeping the ones related to the current location of a mobile user. In MOPSI we use the same mechanism to allow users to subscribe to notifications of actions happening nearby. The GUIDE system [4] was developed to provide city visitors with a hand-held tourist guide. The authors solve the issues that arise from the development and deployment of a context-aware electronic tourist guide in a practical real-world environment: the city of Lancaster. O-MOPSI[11] is a mobile orienteering gaming system which uses the location of entities from MOPSI. A game is defined by a set of goals that players need to reach as fast as possible and in no particular order. Similarly as in [4], we record actions when the player reaches the goals.

In [5], the authors describe a method for inferring user similarity based on similar actions. They find places where users visit by first detecting stopping points in their trajectory and then trying to map the stop to service locations from a separate database. This visit detection system works for old data, collected over a period of time because it relies on finding significantly long stops in user movement. In MOPSI we approach the problem in a different way. Let us consider the following action:

Andrei was at Pullapuoti yesterday.

It can be important to users who might think that it is worth to go there because *Andrei* appears to visit frequently. Our aim is to identify the visits in real-time:

Andrei is at Pullapuoti.

[10] http://cs.uef.fi/mopsi
[11] http://cs.uef.fi/o-mopsi/

Actions detected in real-time are significantly more important. Another user might see that *Andrei* is at *Pullapuoti* and might even consider joining if it is close enough.

JEDI [6] stands for **J**ava **E**vent-based **D**istributed **I**nfrastructure and is an object-oriented infrastructure which supports a flexible interaction among distributed software components. JEDI allows mobile application components to produce notifications by connecting to a logically centralized notification dispatcher that has global knowledge of all subscription requests and user actions. The benefit when using this mechanism is preserving the identity of the users when distributing user actions. In MOPSI, private users can exist, which share information with a subset of all other users. It is essential we also hide their identity from others and show their location to users entitled to see it.

Data loss can also be an issue. In [7], the authors address this problem but only for stationary users. In MOPSI we analyze the feedback from attempts of sending information and upon failure choose to resend the data.

In MOPSI, some user actions are detected by noting user interactions with other users or places. Examples are several users traveling together or a particular user visiting a certain place. *Clustering*[12] is a method for organizing objects into clusters (groups) based on a similarity measure so that objects inside a cluster are similar to each other while dissimilar to the objects in the other clusters. Clustering is solved by various algorithms which can be classified based on the cluster model [8]. MOPSI users are considered to form meetings (groups) when they are within a fixed small distance to each other. We can find these groups by using clustering.

Because the thesis is strictly linked to the MOPSI application, the following section better explains what MOPSI is and what data is available inside the system.

[12] http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/

2 MOPSI Application

MOPSI is a locator assistant that helps individuals to know where their friends are and what is around them. It supports photo sharing, easy tracking, and chatting with friends. MOPSI can be found on the web at http://cs.uef.fi/mopsi and mobile applications for all major platforms can be downloaded from http://cs.uef.fi/mopsi/mobile.php.

MOPSI is developed by the Speech and Image Processing Unit[13], School of Computing in the University of Eastern Finland[14]. The name originates from a Finnish abbreviation for **Mo**biilit **P**aikkatieto**S**ovellukset ja **I**nternet which is translated into English as Mobile Location-based Applications and Internet. In Scandinavian and Slavic languages the pug dog[15] is called mops (mopsi). MOPSI project provides location-based services such as search, recommendation, data collection, bus transportation, users tracking and the relatively new action notifications which are the main subject of the thesis. In the following sub sections the main features of MOPSI are described in more detail.

2.1 Data Collection

In MOPSI two types of user collected data are stored: geo-tagged *photos* and *routes*. Photos may have an associated description which users can type when the photo was taken. Upon photo upload, location (latitude and longitude) is also sent and stored in the database. A route is a sequence of points recorded at a fixed interval. The system distinguishes the following modes of transportation: walk, run, bicycle and car [9].

Figure 4 shows both types of results (routes and photos) for a selected user, *Pasi*. His entire collection until 25.4.2013 has over 800 routes consisting of over 1.5 million points. To display such a massive amount of data the system uses *polygonal approximation* and *bounding box* solutions as described in [10]. *Pasi* also has over 5000 photos. They are placed on the map at the locations they were taken. To avoid overlapping elements, the photos are clustered using a grid-based method[16]. Photos appear also in the timeline view, on the top of the web page, clustered by time.

[13] http://uef.fi/en/sipu
[14] http://uef.fi
[15] http://en.wikipedia.org/wiki/Pug
[16] http://cs.uef.fi/pages/franti/sipu/MSc_Thesis_Chen_Jinhua.pdf

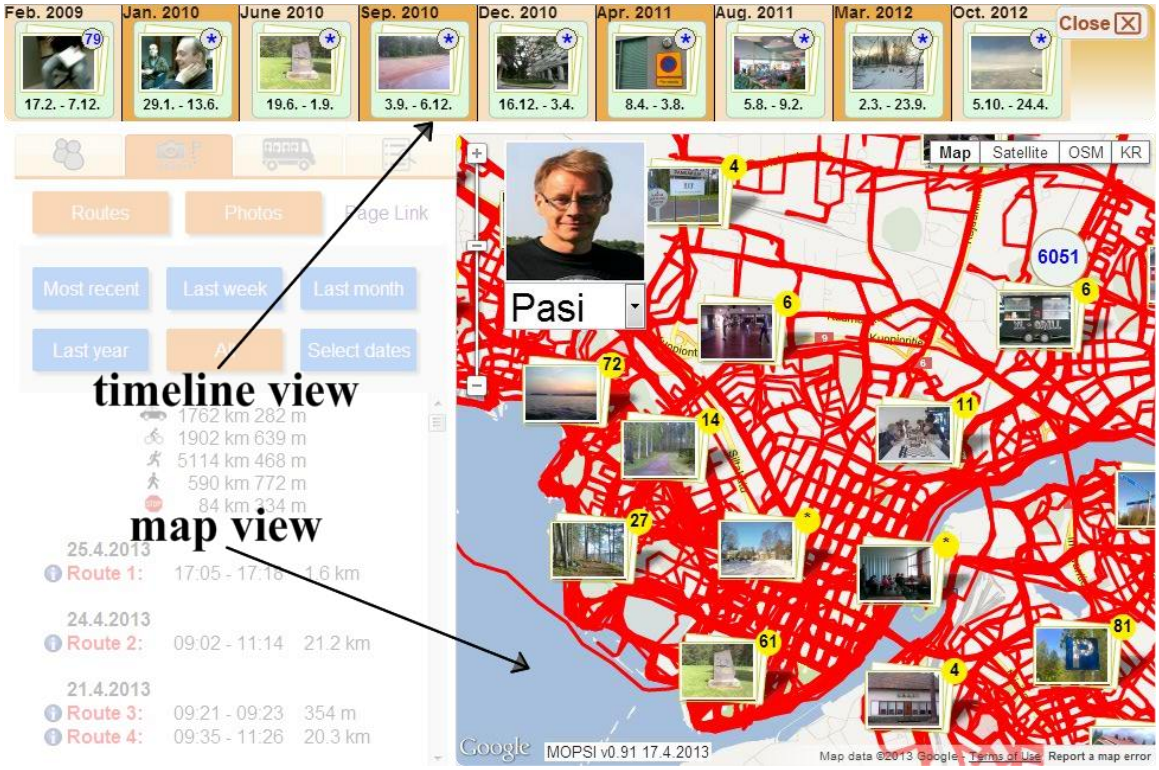


Figure 4: MOPSI on web showing the entire data collection of user *Pasi*.
Map is focused over *Joensuu*

Figure 5: MOPSI on mobile (Windows Phone) showing a photo (left) and a route (right)

The mobile provides a simplified interface for viewing a user's collection. Linear access by time is implemented, most recent collection being shown first. Figure 5 shows how a photo and a route from *Pasi's* collection appear on MOPSI for Windows Phone.

A route can also be analyzed by automatic segmentation and the detection of the movement type is done for each segment [9]. Figure 6 shows a route belonging to user *Radu*. The route is segmented and classified as mostly cycling activity with several stops.

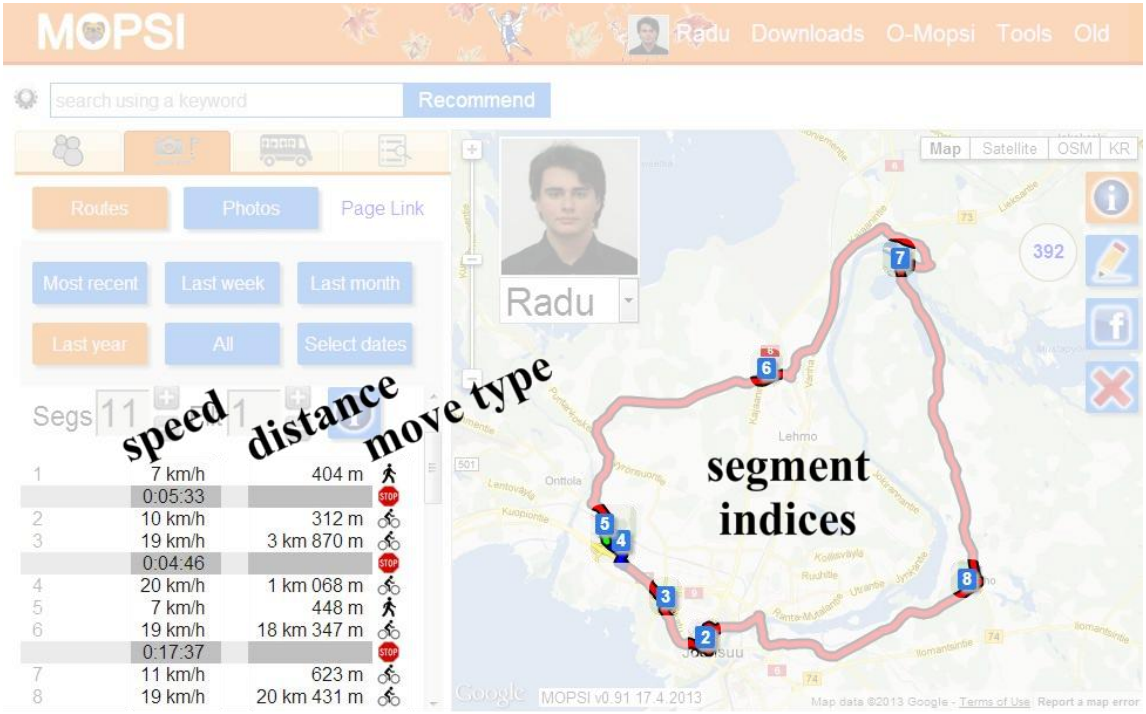


Figure 6: Analyzed route containing mostly cycling activity

2.2 Search

The popular search engines Google, Bing and Yahoo provide fast and relevant information. However, they don't utilize one important aspect of relevance: the location of the user. One reason for this is that location was not as widely available in the past as nowadays when GPS technology is integrated in most smartphones. Another reason is that web pages are rarely attached with location information [11]. The following example shows location inside HTML META tags which are machine parsable.

```
<HTML>
<HEAD profile="http://geotags.com/geo">
<META name="geo.position" content="62.35; 29.44">
<META name="geo.region" content="FI">
<META name="geo.placename" content="Joensuu">
```

Location-based search engines such as Google maps and Yellow pages use databases where the entries have been geo-referenced beforehand. The downside is that the data must be collected from providers (service owners and administrators) which need to put efforts to keep the location information up-to-date [11].

MOPSI search combines the traditional location-based service and search engine. It retrieves data from the local database, then queries relevant data from the user collections and finally performs location-based search from web as originally proposed in [12], and later implemented as summarized in [13]. The key idea is to use ad-hoc geo-referencing of the web pages based on address detection within the body text [14], rather than relying on geo-tags or address tags which rarely exist.

A user may access MOPSI search from the web[17] as illustrated in Figure 7. Location can be selected by dragging the marker to the desired place on map or by typing an address in the appropriate field. Keyword must also be specified. The MOPSI mobile solutions[18] also provide access to the search. Figure 8 shows this feature on Windows Phone. Here location information is automatically detected by the phone. The only needed input by the user is the keyword.

MOPSI search engine allows the user to find nearby *services* and photos from user collection. *Services* represent a variety of categories such as restaurants, bars, cafeterias, grocery stores, museums, pharmacies and ATM machines. They are verified by administrators and are illustrated with green color coding. The information associated with them contains the title, location, photo, description, web link, keywords and user ratings. Photos are part of user collections and are marked with yellow coding.

[17] http://cs.uef.fi/mopsi
[18] http://cs.uef.fi/mopsi/mobile.php

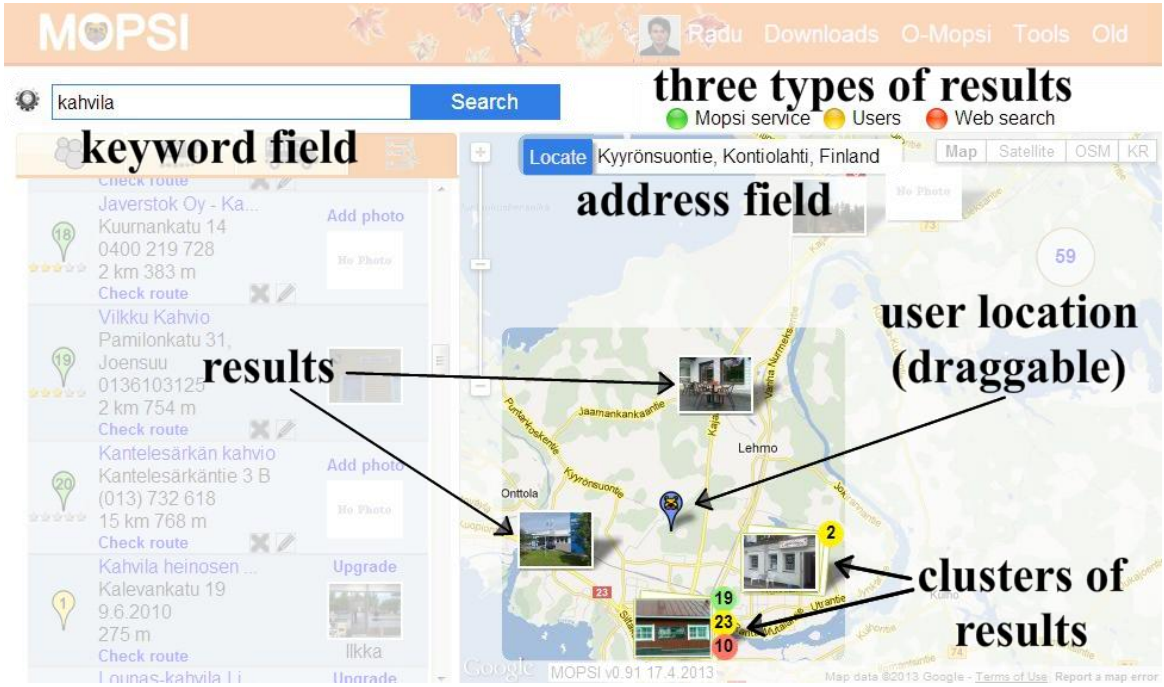


Figure 7: MOPSI web - searching for *kahvila* in *Joensuu*

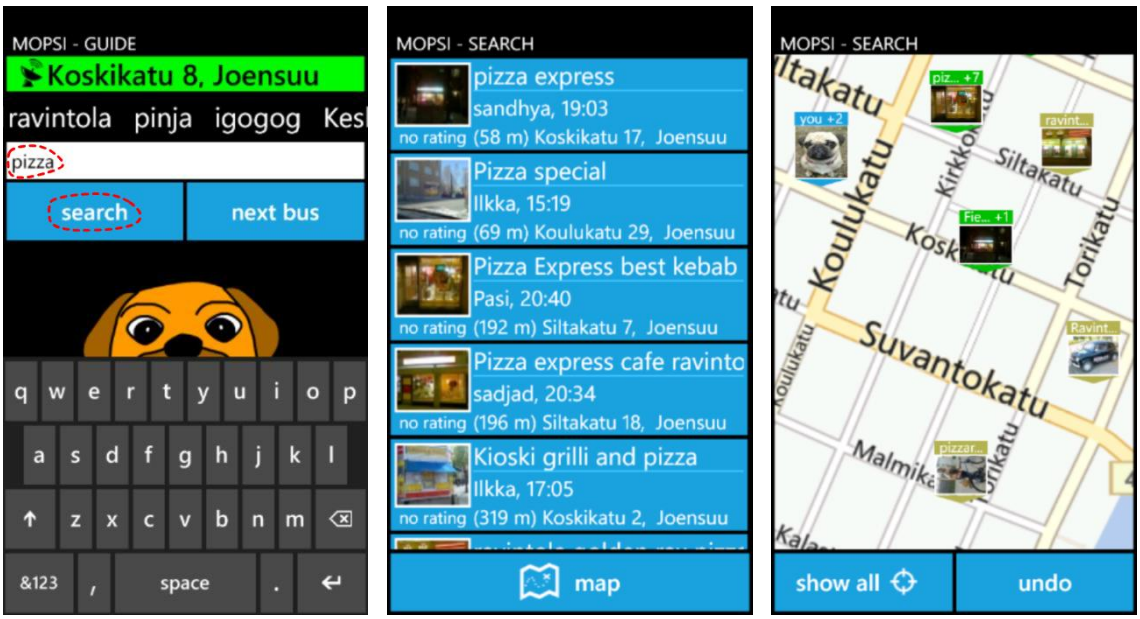


Figure 8: Searching for *pizza* in *Joensuu* using MOPSI for Windows Phone

2.3 Recommendation

Recommendation systems try to predict what the user is interested in and provide personalized search relying on a variety of contextual information. For example, Amazon[19] gives recommendations based on last purchases and user ratings (Figure 9).

Figure 9: Books recommended by Amazon

In MOPSI four aspects of relevance [15] are used: *content*, *time*, *location* and user *social network*. The system recommends trusted services, geo-tagged photos and routes. The goal is to offer personalized recommendations [16] by combining the three different data sources as described in subsection 2.2. Binary search is used by doubling or halving the radius depending if too few or too many results are found. The process continues until enough results are found or until the radius covers the entire planet. A minimum of 10 results is the threshold for stopping the search, see Figure 10.

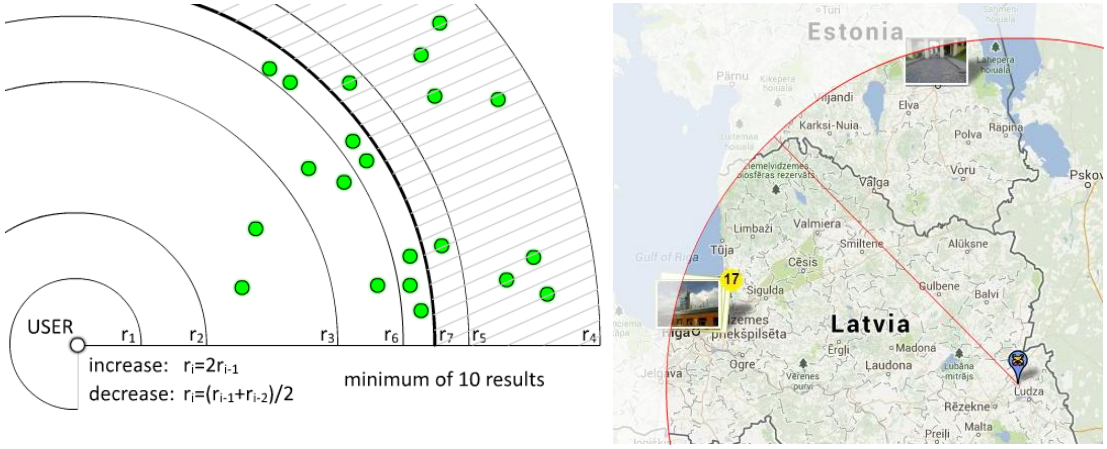


Figure 10: Changing radius to find results when nothing is close

[19] http://www.amazon.com/

A user may ask for recommendations using the MOPSI web page (Figure 11). In addition to time and location, the results also depend on the user profile and automatically obtained preferences. If the user is not logged in the results are the same except user interests which are excluded from the search criteria. Recommendation does not require additional user input, as opposed to the keyword required by the MOPSI search.

A MOPSI user can also use recommendation on the mobile trough a single tap of a button (Figure 12). Location is automatically detected from the GPS sensor. Time and user identity are directly available in the application.

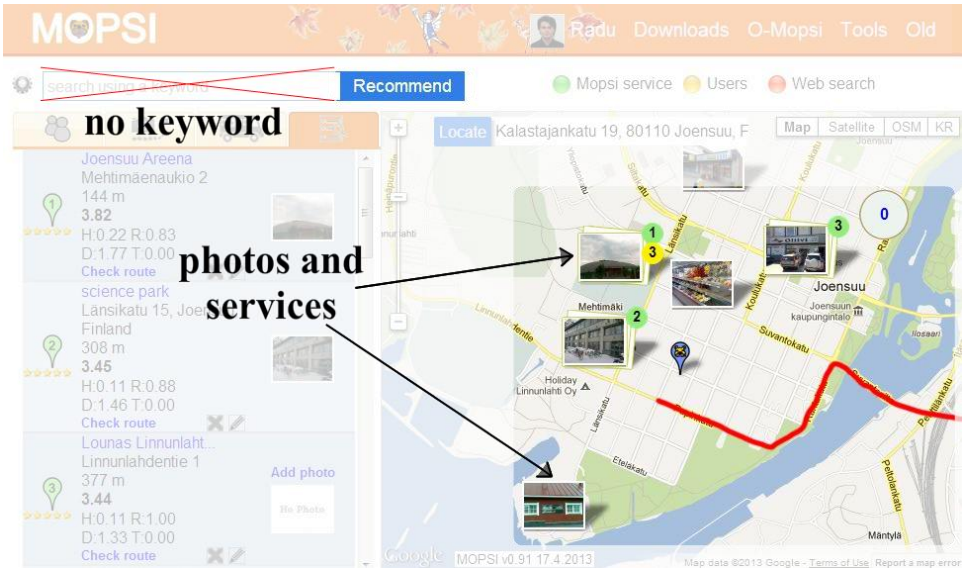


Figure 11: MOPSI web showing recommendations in *Joensuu*

Figure 12: MOPSI for Windows Phone showing recommendations in *Joensuu*

2.4 Users Tracking and Actions

By *users tracking* we refer to the ability to see the locations of other users in real-time. In Figure 13, the users are listed on the left side of the screen sorted according to how recent was their last activity in MOPSI. In addition, three most recent user actions are displayed for each user. On the map, users are clustered shown by a bubble with the name of the most recently active user.

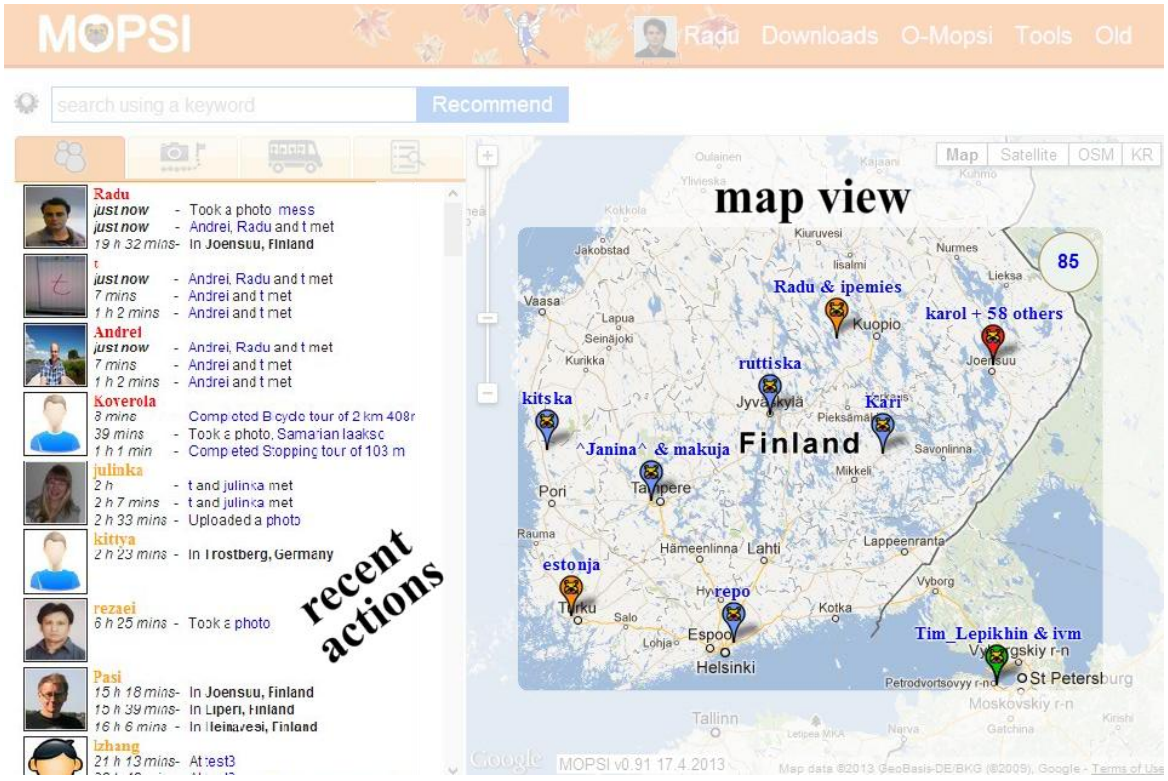


Figure 13: MOPSI web showing users' locations and actions in real-time

The mobile solutions also provide access to this information. Figure 14 shows the user list, clusters on the map and recent user actions on three different pages. This separation is needed because the screen size is significantly smaller on smartphones. The user actions will be discussed more detailed in section 3.

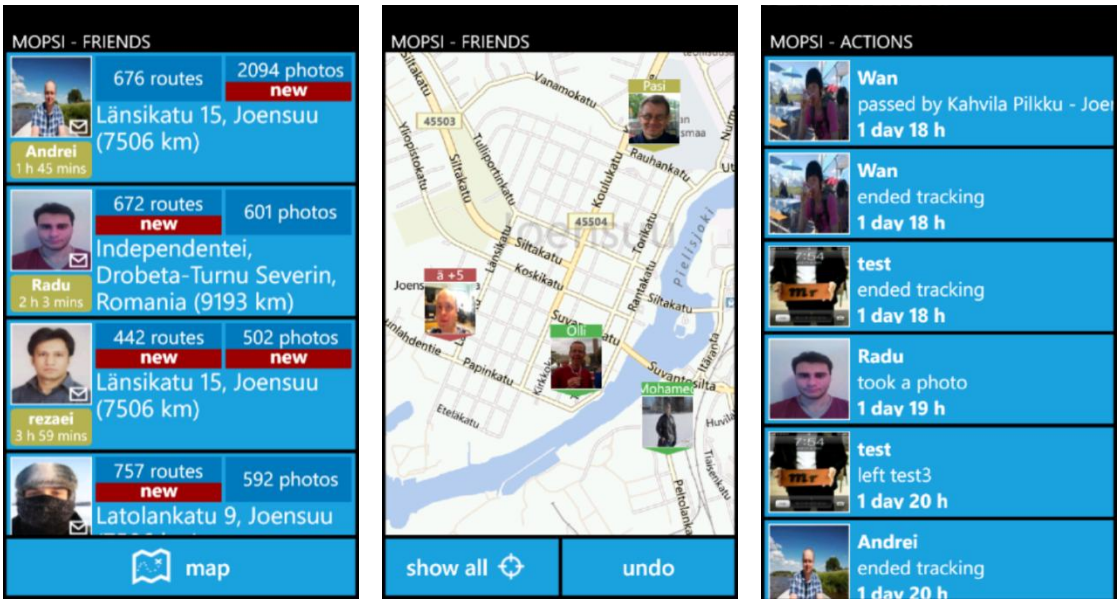


Figure 14: Windows Phone MOPSI showing (from left to right): friends list, friends clustered on the map and recent actions

3 User Actions

In addition to the data (routes and photos), other user actions are also recorded such as: login / logout, completing tracking, taking / uploading photos, changing city, visiting, passing by or leaving a service, adding a service and meetings between users.

We classify these user actions in two categories:

- Triggered actions;
- Concluded actions.

Triggered actions are recognized upon a single HTTP request made to the server by the mobile device of a user. They do not require additional system information to be recognized. User actions that fit in this category are:

- Login;
- Logout;
- Taking / uploading photos;
- Completing tracking;
- Adding a service.

Concluded actions need additional system information when being generated. Similarly as in [17] the state of the system at different past moments is needed when making these conclusions. Therefore we store this information and use it when recognizing these actions. We check if actions can be concluded periodically (every 30 seconds). User actions that fit in this category are:

- Logout (due to missing connection with the server);
- Changing city;
- Visiting a place;
- Passing by a place;
- Leaving a place;
- Meeting between users.

In the following subsections each action is discussed in detail. We describe the methods and reason why they are most suitable for the MOPSI environment. Time and memory complexities are analyzed when relevant.

3.1 Login and Logout

Anybody can start using MOPSI by first creating an account. This can be done from the web page or any mobile application. A user logs in to the mobile version of MOPSI by entering valid credentials. The user logs out by pressing the logout button or by closing the application. These actions are *triggered* directly by the user. However, for determining the status of being online/offline this method alone is not sufficient mainly because of connection issues.

Missing connection between the mobile device and the server can happen because of several reasons including:

- *Network specific*: such as internet connection time-out or signal fluctuations;
- *Device specific*: such as battery loss or firmware error;
- *Application specific*: crashing or malfunctioning;
- *Server specific*: if the server is closed, busy or system error occurs.

Keep-alive mechanisms (Figure 15) send signals (messages) to a server at fixed time intervals. The server expects these signals from all connected users. Typically on the server a user is marked as offline if no signal is received from that particular user.

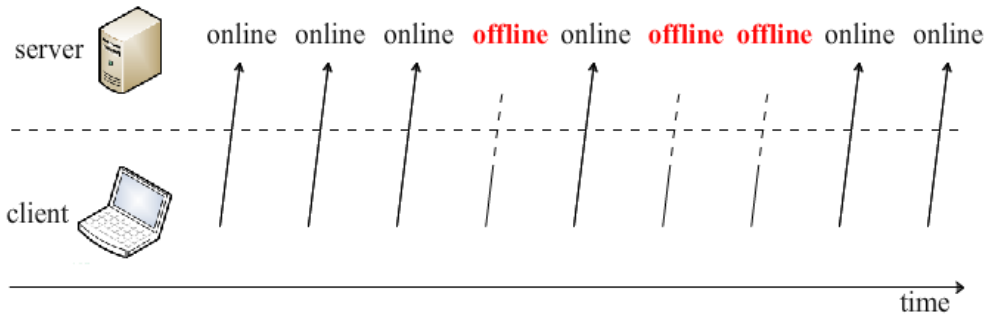


Figure 15: Keep-alive mechanism

In MOPSI we use a keep-alive mechanism to *conclude* logout actions. The devices send the signal to the server every 30 seconds. If a user fails to signal, it is marked as idle (still appears to be online to other users). After two minutes, an idle user status is set to offline. Figure 16 illustrates how the connection is evaluated in time.

Because mobile devices have a tendency to lose connection for brief periods of time, we concluded two minutes of inactivity to be sufficient to logout a user. Smaller values yield frequent changes of the online status and larger makes user to appear (in the idle state)

longer, even though he or she cannot be reached. Both cases can cause confusion for the users.

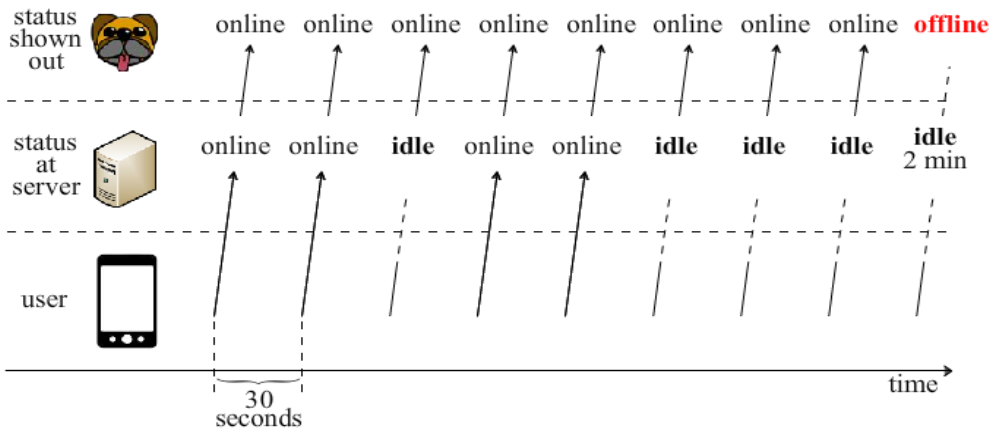


Figure 16: Concluding login status in MOPSI

3.2 Taking and Uploading Photo

Because of issues with server connectivity, in MOPSI we differentiate two separate actions when a photo reaches the server similarly as in [18]:

- Photo *taking*;
- Photo *uploading*.

We want to avoid confusion such as:

Pasi published a photo. It's from Kenya! But I saw him just this morning in Joensuu...

We also want to stress the importance of a photo being taken right now:

Mohammad took a photo? He's back! Maybe we can play football in the afternoon.
Karol took a photo? Ah I see, he's at Koli, I know that place!
Andrei uploaded 5 photos? He probably shares holiday pictures. I'll watch them later.

Both actions are *triggered* in the same way, when uploading of a photo to the server completes. The difference is made by analyzing the difference between two timestamp values:

- *Photo timestamp*, the time the photo was taken;
- *Upload timestamp*, when the photo is successfully stored in server.

Due to connectivity problems or the usage of offline mode, photos may be buffered a significant amount of time before they are sent to the server. When the photo reaches the server, the two timestamps are compared. If the difference is smaller or equal to a threshold (10 minutes in MOPSI), the action of taking a photo is recorded. If it is not the case, the photo upload action is recognized (Figure 17).

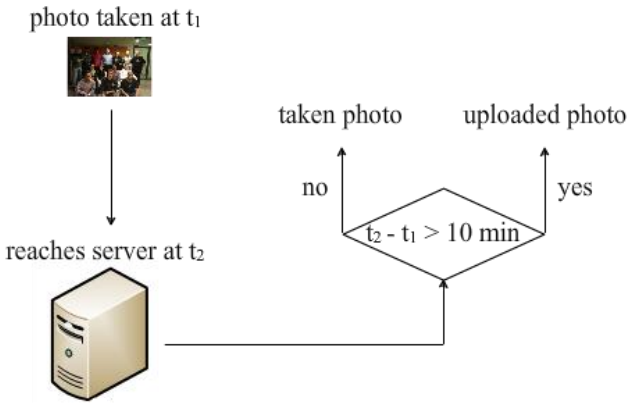


Figure 17: Decision between taking a photo and uploading a photo

3.3 Completing Tracking

Tracking refers to recording a traveled path. MOPSI users can perform tracking so that points are recorded at fixed time intervals, which can be chosen as a parameter in the settings. Default value is 4s (seconds) but 1s or 2s can provide better accuracy. Higher values are not meaningful because the route reduction procedure [19] can decrease the amount of data while keeping the accuracy in a given threshold.

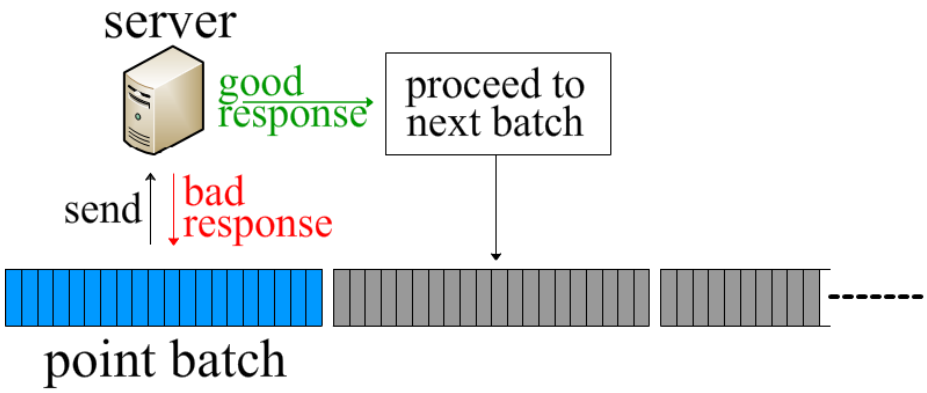


Figure 18: Point batch buffering

Tracking in MOPSI is real-time. An ideal system would behave so that any new point is immediately sent to the server. Because of the possibility of connection loss with the

server, buffering (Figure 18) is implemented on the mobile device. This ensures that no points are lost during the upload process and also sending points individually would overwhelm the network, which is the biggest reason for battery running out. Because of this new points are sent in fixed sized batches (20 points each). When a user turns off tracking on the mobile device, the remaining points are sent in one batch. This batch is marked as the final and when it arrives on the server it completes the route.

Movement type analysis [9] is then made on the complete route. The *tracking completed* user action is *triggered*. Here are some examples of corresponding notations:

Oili completed Walking tour of 7 km 204 m.
Pasi completed Running tour of 27 km 686 m.

3.3.1 Move type detection

The first challenge of the movement type detection is to split the route into several segments based on similar speed. The second part is to classify the segments in one of the 5 move types: *stop*, *walk*, *run*, *bicycle* or *car*.

The algorithm divides the route into segments with similar speed while automatically calculating the number of segments. Input is a route $\boldsymbol{R} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \dots, \boldsymbol{p}_n)$, where $\boldsymbol{p}_i = (x_i, y_i, t_i)$, and the corresponding speeds are $(v_1, v_2, \dots, v_{n-1})$. For a given segment number *m*, we define a cost function that minimizes the sum of the inner speed variance in all the segments:

$$f = \sum_j \sigma_{i_j}^{i_{j+1}} \left(t_{i_{j+1}} - t_{i_j} \right) \quad (1)$$

where i_j and i_{j+1} are the indexes of the start and end points of the segment *j*, and σ is the speed variance between the points *j* to *j+1* in route $\boldsymbol{R}$.

This minimization process is solved by a dynamic programming process in $O(n^2 m)$ time and $O(nm)$ space, where the speed variance can be calculated in $O(1)$ time by using pre-calculated accumulated sums. Optimization is done in the $n \times m$ state space using dynamic programming as follows:

$$D(s,r) = \min\left(D(c,r-1) + \sigma_c^s(t_s - t_c)\right), c \in (1 \dots s-1)$$
$$A(s,r) = \arg min_c (D(c,r-1) + \sigma_c^s(t_s - t_c) \quad (2)$$

where $s = 0 \dots n$, $r = 0 \dots m$, with an initial condition $D(0,0) = 0$, and $A(s,r)$ is the index for backtracking. The number of segments m_0 is determined by

$$m_0 = \arg min_i\big(D(n,i) + \lambda_1 i + \lambda_2(t_n - t_1)\big), i = 1 \dots m \quad (3)$$

where λ_1, λ_2 are regularization parameters.

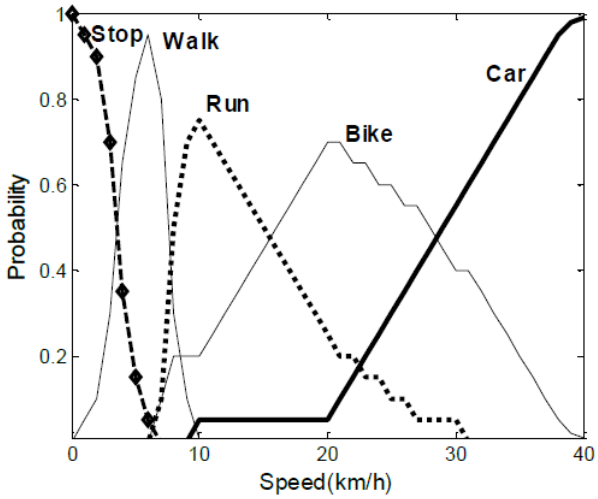


Figure 19: A priori probabilities for soft classification of the route segments

After we have the segments we perform soft classification of each segment as *stop*, *walk*, *run*, *bicycle* or *car* using the a priori probabilities shown in Figure 19. A second order hidden Markov model (HMM) is used where the hidden states represent the movement types and the observed data are the features for each segment. Correlations are made both to the previous and the next segment.

For the cost function of the 2nd order HMM we use:

$$f = \prod_{i=1}^{M} P(m_i|X_i, m_{i-1}, m_{i+1}) \quad (4)$$

where m_i={*stop*, *walk*, *run*, *bicycle* or *car*} in the state of segment *i*, *Xi* is its feature vector, m_{i-1}, m_{i+1} are the states of the previous and the next segment. The probability that a segment would have a hidden state m_i depends on the previous state, the next state and its feature vector. After Equation 4 has been maximized, the most likely sequence of the hidden state $m_0, m_1, \dots, m_M$ is determined.

Assuming the feature vector X_i is uncorrelated with m_{i-1} and m_{i+1}, this cost function can be concerted by applying Bayesian inference:

$$f = \prod_{i=1}^{M} \frac{P(m_{i+2}|m_i, m_{i+1})P(m_{i+1}|X)}{P(m_{i+2})} \quad (5)$$

where $P(m_{i+2}|m_i, m_{i+1})$, $P(m_i|X_i)$ and $P(m_i)$ are all prior information. In the implementation of the algorithm, dynamic programming is employed for maximizing the cost function (5).

3.3.2 Novelty estimation

We have designed methods for calculating *route similarity* and estimating the *novelty* of a route. The similarity of two routes is defined as the number of intersecting points relative to the points in one route:

$$s(R_a, R_b) = \frac{|R_a \cap R_b|}{|R_a|} \quad (6)$$

where route R_x is represented as a set of points $\{p \mid p \in R_x\}$.

Formula 6 typically gives different outcome if the parameters are reversed $s(R_a, R_b) \neq s(R_b, R_a)$. The only case $s(R_a, R_b) = s(R_b, R_a)$ is possible when the two routes have equal number of points $|R_a| = |R_b|$. Two points are considered a match if the distance between them is less than a threshold, currently set to 50 meters. Distances between points on the earth's surface are calculated using the *haversine* distance

$$h(p_1, p_2) = r \cdot 2 \arcsin\left(\sqrt{sin^2\left(\frac{\Delta\phi}{2}\right) + cos\phi_1 cos\phi_2 sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \quad (7)$$

where

- ϕ_1, ϕ_2 are the latitudes of p_1 and p_2 respectively;
- λ_1, λ_2 are the longitudes of p_1 and p_2 respectively;
- r is the radius of the earth (~6373 km).

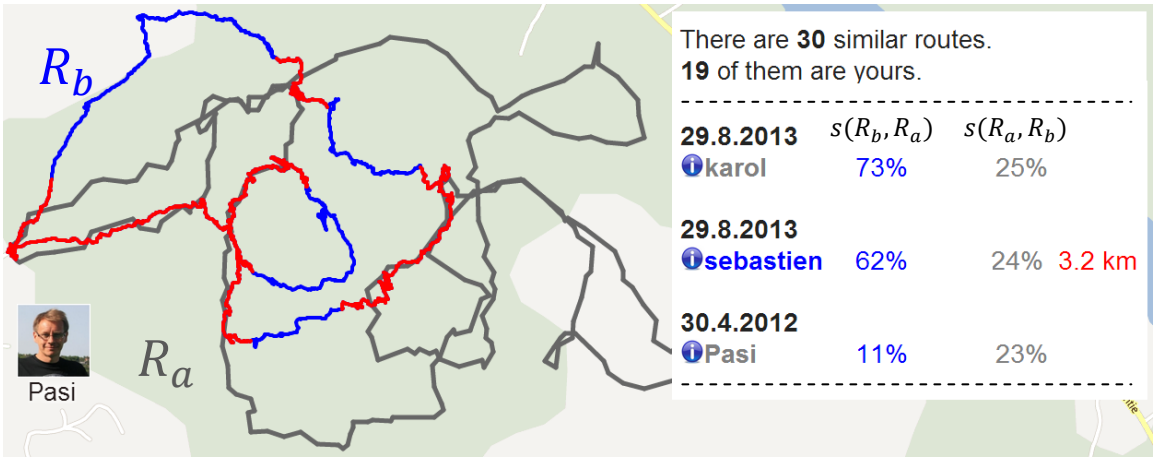


Figure 20: *Pasi*'s route (gray) is matched to *Sebastien*'s route
Red represents the matched and blue non-matched part

In Figure 20, a sample route R_a from *Pasi*'s collection (recorded on 29.8.2013) is taken as a reference, 30 similar routes exist 19 which belong to *Pasi* and 11 to others. One of the similar routes belongs to user *Sebastien* R_b. The similarity of *Sebastien*'s route with respect to *Pasi*'s route is 62%, meaning that 62% of the points in *Sebastien*'s route are mapped to *Pasi*'s route $s(R_b, R_a) = 0.62$ (red segments with a total length of 3.2 km). Matching the other way around we see that *Pasi*'s route is only 24% similar to *Sebastien*'s route $s(R_a, R_b) = 0.24$. In Table 1 all similar routes to the reference route are shown.

Table 1: Routes similar to Pasi's from 29.8.2013.

Date	Time	User	$s(R_b, R_a)$	$s(R_a, R_b)$
14.8.2013	17:30-18:56	**Pasi**	**73%**	**54%**
20.6.2013	18:05-19:56	Pasi	59%	54%
20.6.2013	18:05-19:56	chait	59%	54%
26.5.2011	17:57-19:24	Pasi	55%	53%
8.9.2011	19:07-20:18	Pasi	73%	43%
3.9.2009	18:05-19:09	Pasi	48%	40%
7.6.2012	17:49-18:49	Pasi	63%	33%
29.8.2013	17:20-18:06	karol	73%	25%
29.8.2013	18:12-19:52	**sebastien**	**62%**	**24%**
30.4.2012	16:04-18:08	Pasi	11%	23%
24.3.2013	13:18-15:04	Pasi	18%	22%
2.1.2011	13:40-15:19	karol	17%	22%
30.7.2011	11:47-14:56	Pasi	11%	21%
7.6.2012	17:40-19:48	Low2	31%	21%
24.8.2013	15:15-17:47	Pasi	8%	18%
8.9.2011	18:21-20:23	Low2	51%	18%
8.9.2012	09:51-13:01	Pasi	6%	17%
11.8.2012	11:27-14:39	Pasi	7%	17%
25.6.2010	09:16-11:23	Pasi	10%	16%
21.7.2012	14:39-17:12	Pasi	6%	12%
24.6.2013	15:03-15:04	chait	52%	11%
11.7.2010	10:04-12:01	karol	11%	9%
22.5.2010	09:29-11:07	Pasi	4%	6%
6.4.2013	18:12-18:17	matti	76%	5%
16.8.2012	17:59-18:27	Pasi	35%	5%
27.7.2011	11:22-13:36	Pasi	2%	5%
27.7.2011	12:31-13:36	Low2	5%	4%
4.6.2011	08:29-11:32	Pasi	1%	4%
6.4.2013	18:34-18:40	matti	10%	1%
10.9.2011	11:19-12:48	Pasi	1%	1%

Using the route similarity we next define *novelty*, to estimate how original a given route is. We calculate the novelty of a route with the following formula:

$$Novelty(R_a) = 1 - \max_{i=\overline{1,N}, i \neq a} s(R_i, R_a) \quad (8)$$

where N is the total number of routes in the system.

The most similar route to *Pasi*'s reference route belongs to him also, see Figure 21. The novelty is 46%, meaning that at most 54% of the route can be matched to another route from the database. If a user records a route in a location where no other MOPSI user has been, it is 100% novel.

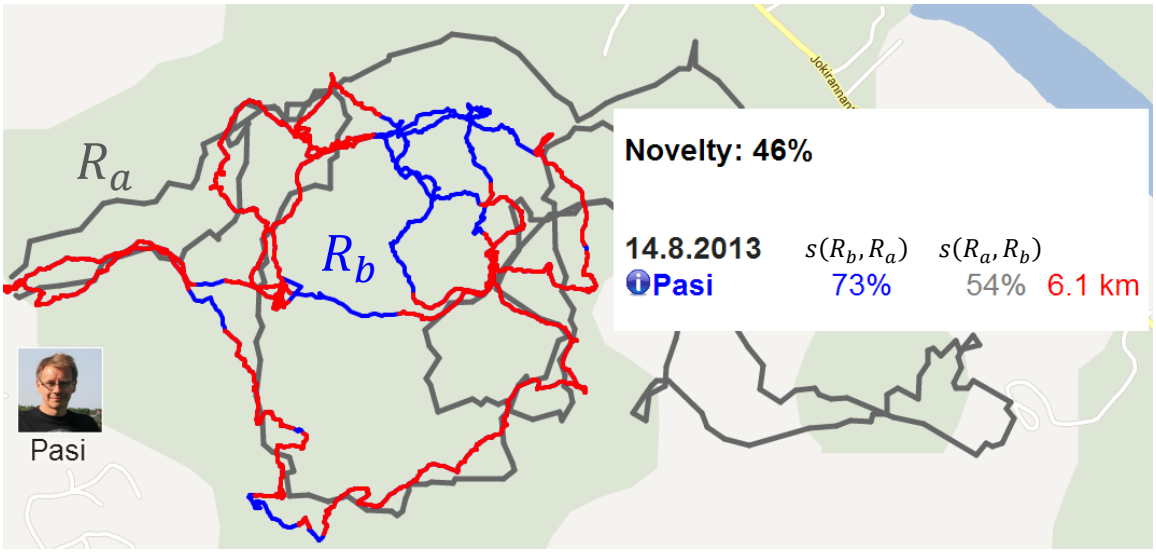


Figure 21: Most Similar route and Novelty

At the moment we cannot calculate the similarity in real-time. The implementation relies on a cron job which runs every midnight and calculates the similarity for the new routes recorded during the previous day. The algorithm compares the new routes (M) with all other routes in the database (N). For each comparison the distance between every point of a route to every point of all other routes is calculated using Formula 7. This task takes $O(M \times N \times P^2)$ time, where P is the average number of points in a route.

In MOPSI, an average of M=8 routes have been recorded daily during 1.12013 – 20.9.2013. On 20.9.2013 approximately N=9000 routes exist in the system with an average of P=750 points per route, meaning a total of 40,500,000,000 distance calculations performed daily. Even if we avoid checking similarity between routes whose bounding boxes do not intersect the complexity decreases only slightly, and is far from being real-time. The novelty calculation is therefore not used in the MOPSI actions currently.

3.4 Creating Service

MOPSI *services* contain a variety of categories such as restaurants, bars, cafeterias, grocery stores, museums, pharmacies and ATM machines. They are created by any MOPSI user but need to be approved by system administrators so that we can guarantee correct information.

One can create a MOPSI service (from the web page) in two ways as shown in Figure 22. The first way is to move the user marker at a specific location on a map and clicking on it. An information window appears with the option to add a service. The second way is to upgrade a photo from the user collection to a service.

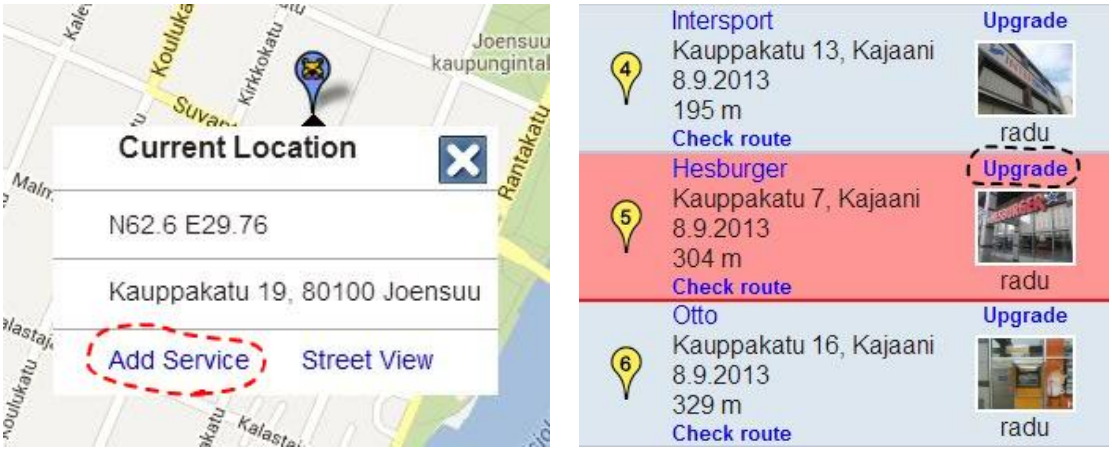


Figure 22: Adding a service at specific location (left), upgrading photo to service (right)

Figure 23: Window for adding service information

In both cases, clicking the *Add Service* and the *Upgrade* buttons, the window in Figure 23 opens and the user fills in the relevant information associated with the service. The difference is that by clicking the *Upgrade* button the title, location and photo are automatically obtained from the user collection. Only the web link, keywords and description (optional) need to be added.

A service is created when the admin user presses the *Save* button. The job of administrators is to periodically check the newly added services to verify that the information is correct and then confirm the service using the interface shown in Figure 24.

Index	ID	Service ▾	Author	Date	Status
259	268	Suvas - ylioppilaskunnan sauna	Pasi	old	Confirmed
258	269	Skarppi - ylioppilaskunnan sauna	Pasi	old	Confirmed
72	5415	Likolampi Camping	Pasi	old	Confirm
73	5414	Gröna Lund	Pasi	old	Confirm
74	5411	Kunnonpaikka	Pasi	old	Confirm
75	5409	Lomakeskus Huhmari	Pasi	old	Confirm

Figure 24: Interface for confirming services

When a service is confirmed, the *creating service* user action is *triggered*. This action contains the service name and the name of the user who created it. For example:

Radu created service Ranch.

3.5 Changing City

Changing City happens when users travel. It can have relevance to other users in the following ways:

Sandy is in my home town! Will let her know what is worth to visit there!
Matti arrived in Joensuu! Let's see if he's free in the evening…

To find out the city name we use the geo-coding API from Google[20]. The API offers a limited number of requests (2500) per day, which works unless there are an overwhelming number of users traveling simultaneously. Because of this we try to use rarely, only for users that move a considerable distance.

Alternative solution is to use a free mapping service such as Open Street Maps[21] or purchase a professional package from Google. For Finland, we use own database obtained from the Digiroad service[22].

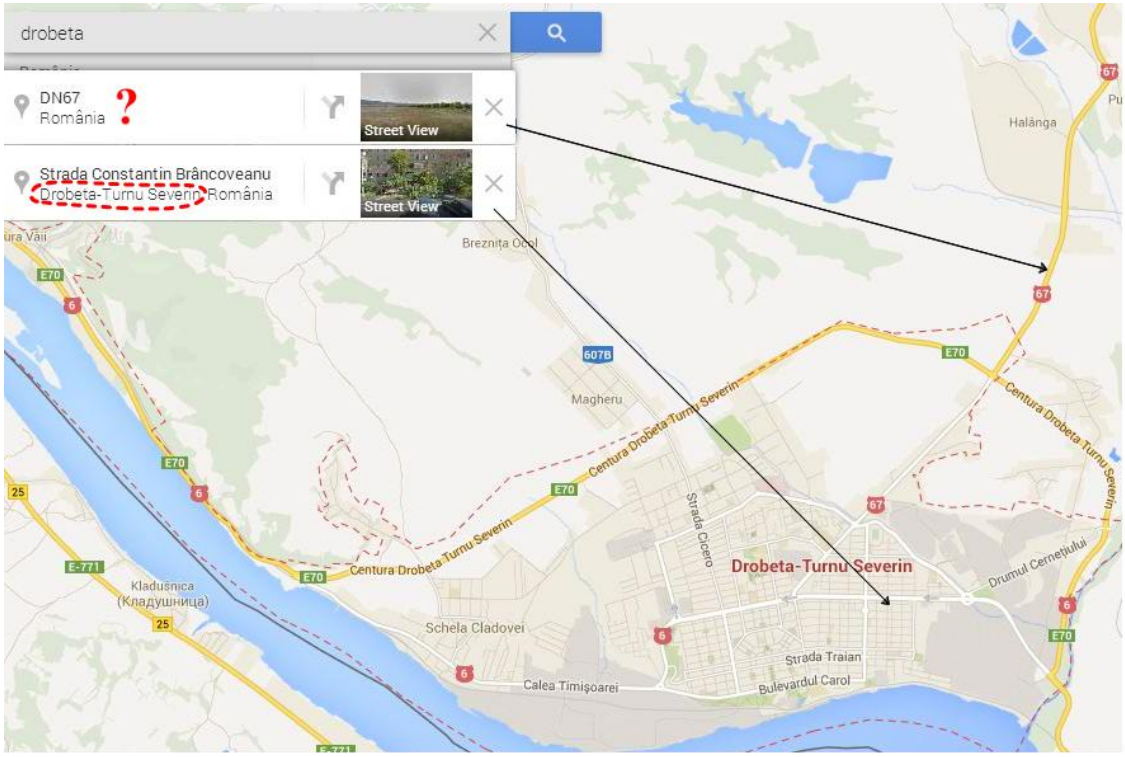


Figure 25: Google maps showing information outside Drobeta-Turnu Severin

[20] https://developers.google.com/maps/documentation/geocoding
[21] http://www.openstreetmap.org
[22] http://www.digiroad.fi

Different countries use different policies to define city borders. For example, in Romania rural areas are not part of any city, whereas Finland is completely divided into municipals that cover the entire country. When the geo-coder does not return any city name (Figure 25), we therefore assume the user is in a rural area.

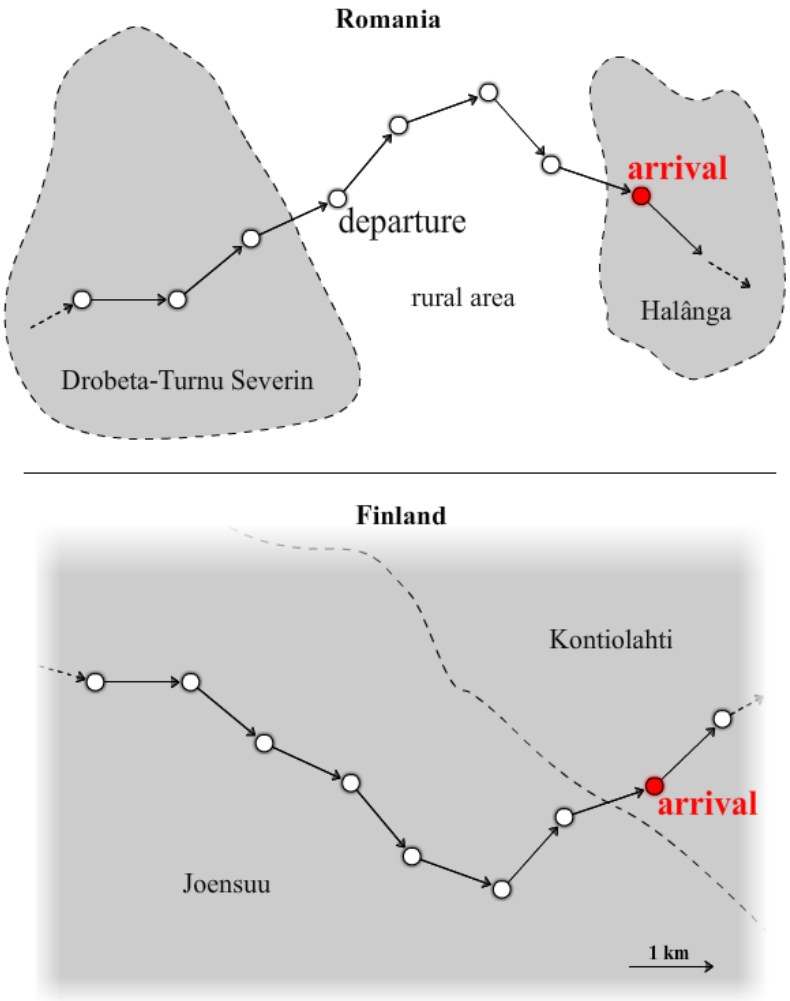


Figure 26: Situations encountered when traveling

To detect that a MOPSI user changes the city, we first select a subset of the user collection, by excluding offline and stationary users. An exception is new MOPSI users for which the action is recorded on first use of the application.

A mobile user makes one request to the Google geo-coder per traveled kilometer. The city name is then compared to the previously stored value. When the values are different the user has arrived in some city and the city changed action is *concluded* (Figure 26).

3.6 Visiting Places

In MOPSI the services represent a variety of categories such as restaurants, bars, cafeterias, grocery stores, museums, pharmacies and ATM machines. They are physical places that can be visited. Three action types can be *concluded* when a user comes in contact with a place:

- *Visit*: when staying at the respective place for a considerable amount of time;
- *Leave*: when moving away from a place the user was previously visiting;
- *Pass-by*: when the user is near the service for a short time before moving away.

Detecting these scenarios is not trivial due to inaccuracy in GPS signal which can make it look like a user is moving away from a place he/she actually continues to visit. We solve this problem by using the *link method* described in the next subsection.

The location of a service is represented by a point in the database. To conclude any of the above actions we need to know which service is nearest to a user's location. To do this we perform two steps:

- Database query for a list of near services;
- Search for the service in the list with minimum distance to the user.

The SQL query is used to find the services inside a bounding box relative to a user location:

```
SELECT id, latitude, longitude FROM services
WHERE  `latitude`  > Sbound
AND    `latitude`  < Nbound
AND    `longitude` > Wbound
AND    `longitude` < Ebound
```

The S, N, W and E bounds are ε meters away from the user location. The query response is a set of nearby services. This process is shown in Figure 27 where the set contains s_1, s_2, s_3 and s_4. The *near* services are then found by calculating distances (Formula 7) to each service in the resulting set and choosing the service with distance below a ε.

We consider a service near a user if the distance is less than ε. In MOPSI, we set $\varepsilon = 25$ meters. A smaller value would cause miss-detection when GPS signal is inaccurate whereas a greater value tends to increase the number of falsely detected actions. In Figure 27, services near to user u are s_1, s_2 and s_3. The smallest distance is then found to be the distance to s_3, which is concluded as the nearest.

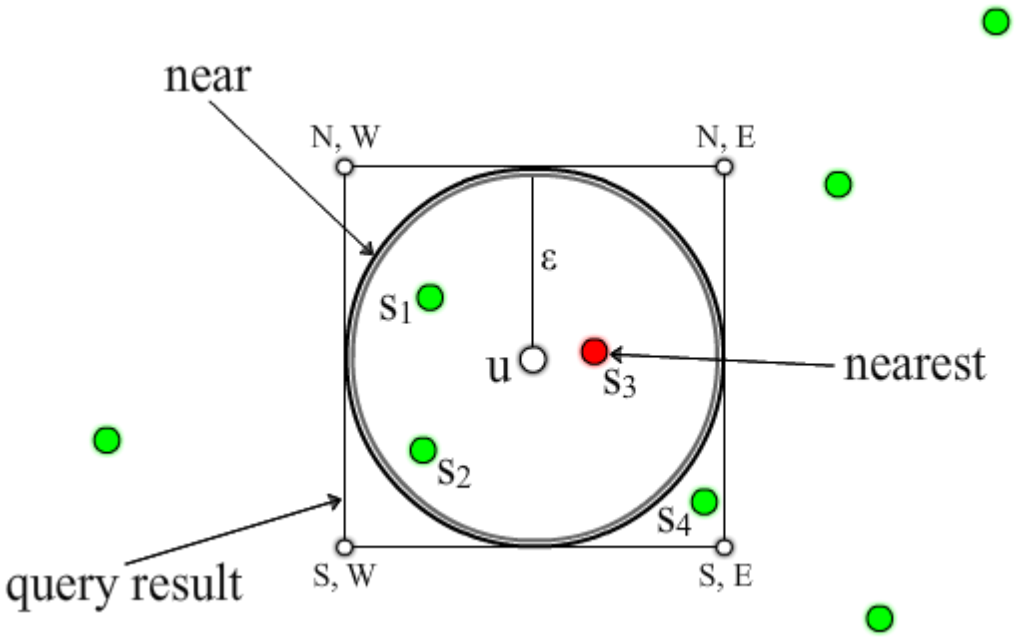


Figure 27: Services near to the user (s_1, s_2 and s_3), nearest service is s_3

3.6.1 Link Method

Because of fluctuations in the GPS accuracy, a user location can slightly change even when standing still. As a result, on the server it may appear that a user is getting farther or closer to the location of a service when the user is not actually nearby at all.

A *link* is a virtual connection established between a user and the nearest service. The goal of defining the link is to deal with the effects of GPS data. To this end, we associate a *strength* value (from 0 to 10) to the link. When the link is formed, the strength is neutral (initialized to 5) allowing it to change in both directions. In the course of time, the strength value increases if the service remains nearest to the user and decreases otherwise.

Figure 28 shows the way a link is being established and how the strength changes in time. At t_3 the user is nearest to service *s* and a link of strength 5 is formed. The strength then increases and reaches the value of 7 at t_5. Then, the link becomes progressively weaker because the user gets farther from the service. At t_{12} the user is *unlinked* from the service when the strength reaches 0. Since the time difference between two successive timestamps Δt is 30 seconds, the link in the example existed for a total time of 4 minutes.

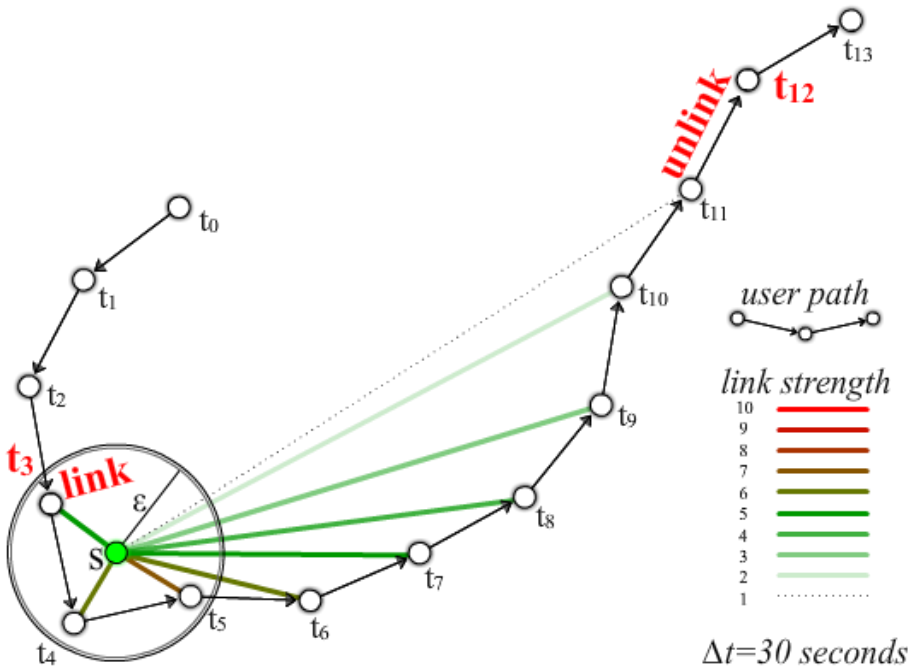


Figure 28: User-to-Service link

3.6.2 Visiting and Leaving

A *visit* is *concluded* when the link strength reaches 10. This corresponds to the user staying at the location of the service for a considerable amount of time (typically around 3 minutes, depending on user movement and GPS accuracy).

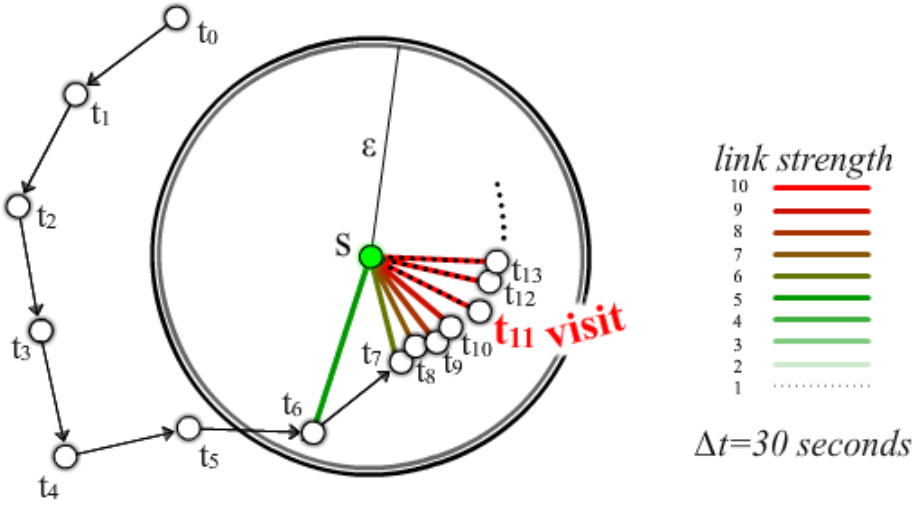


Figure 29: Visit detection

Figure 29 shows a link created at t_6. The strength of the link increases and reaches the maximum at t_{11} where the link strength becomes 10. At this moment, the link is removed and a visit is formed.

Similarly to the link, the visit has an associated strength value, however, it only spans from 0 to 5 and is initialized with the maximum value. We consider the visit strong as soon as it is created because of the confidence given by the previously existing link strength. The strength of the visit varies in time like the link strength: at each time step it increases if the service is nearest to the user and decreases otherwise. If the strength reaches the value of 0, the visit will end and a *leave* action is concluded.

Figure 30 shows how the visit, detected at t_{11}, maintains strength of 5 until t_{55} and then it starts to decrease. The strength reaches 0 at t_{59} concluding a visit that lasted 24 minutes. Here, the *leave* action is detected.

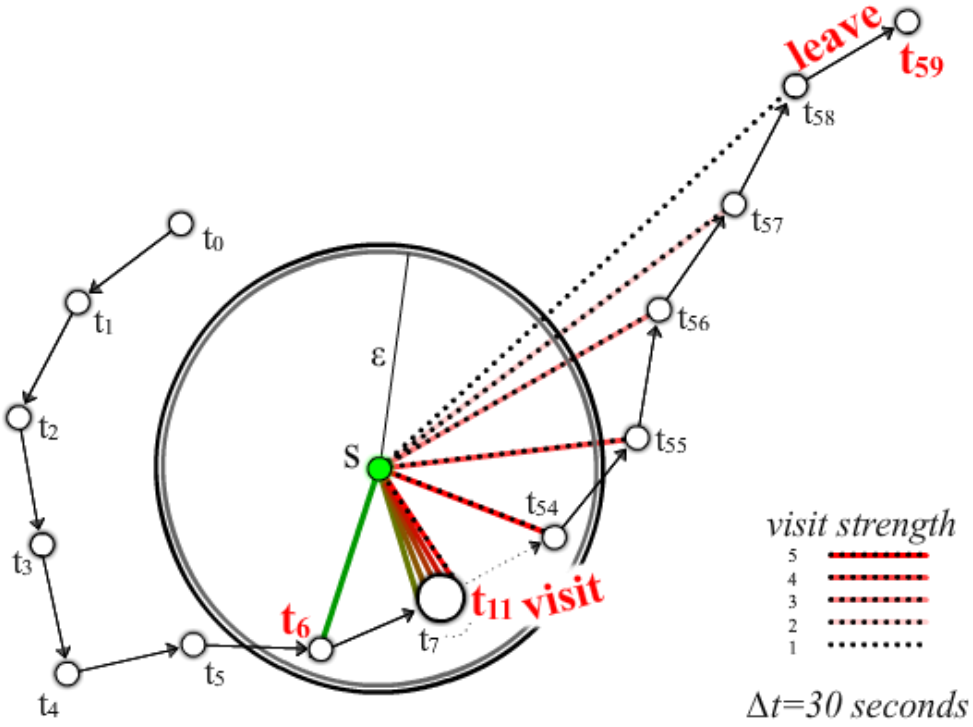


Figure 30: Leaving a service

Figure 31 shows how we detect a visit using the link method despite bad GPS accuracy. At t_6 a link is formed between the user and service *s*. The link strength continues to increase, however, at t_{10} the user location is outside the threshold, therefore, *s* is not nearest to the user and the link weakens.

Without using the link method, at t_{10} it may look like the user is leaving the service. That would be a wrong assumption because analyzing the following locations (at t_{11}, t_{12} …) we realize that the location at t_{10} was probably inaccurate. We note the visit is detected at t_{13}. Two inaccurate location updates (t_{14} and t_{15}) slightly decrease the visit strength, however, they are not enough to terminate the visit and *conclude* the leave action.

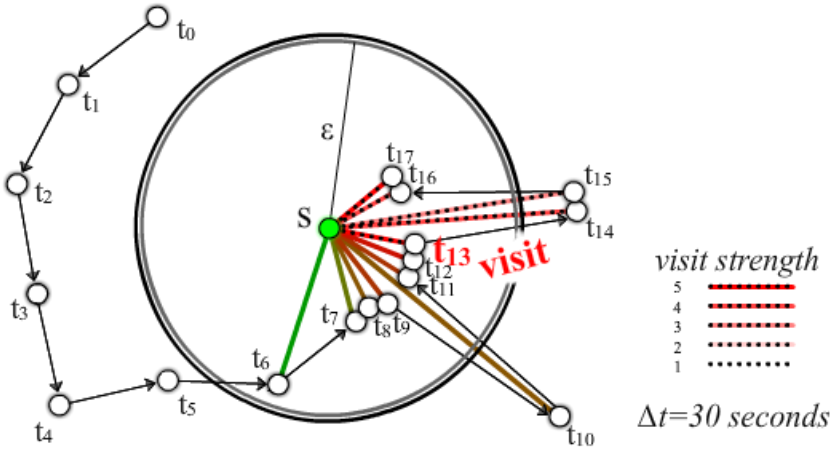


Figure 31: Flexibility of the link method in handling bad GPS accuracy

3.6.3 Passing by

The *pass by* user action is *concluded* when a user goes near a service without staying nearby long enough so that visit cannot be concluded. Using the link method, the pass by action is detected when the link strength reaches 0. Figure 32 shows a link being established at t_1 and a pass by concluded at t_6. At that moment the link is removed.

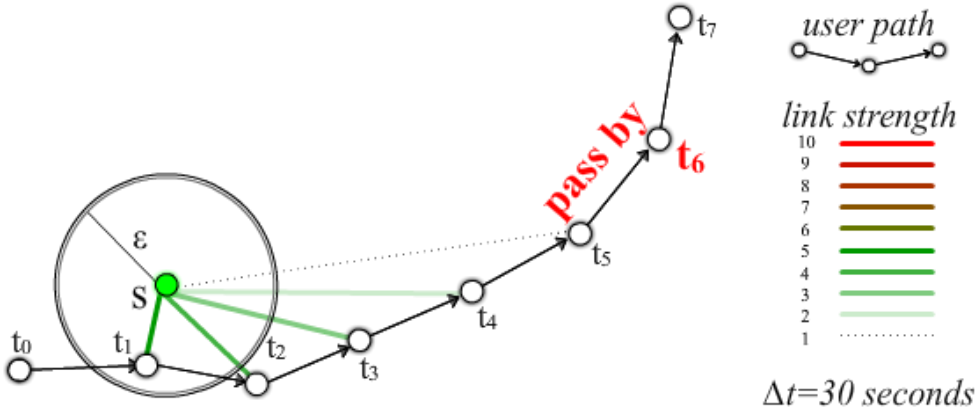


Figure 32: Passing by a service

3.7 Users meeting

By *meeting* we mean multiple users being in close proximity to each other, in other words, part of a group. The concept does not require the participants to be stationary. Meetings can happen when users are standing still, walking, riding bicycles, or traveling by any motorized vehicle. A meeting can be formed by a minimum of two users and does not restrict the number of participants.

To detect meetings, a similar method is used as when detecting visits. The difference is that while a user can only visit a single service at a time, the user can meet several people at once. Therefore we need to change the *link* definition. *User links* are formed between a user and *all other* nearby users as opposed to the links between a user and a single (nearest) service.

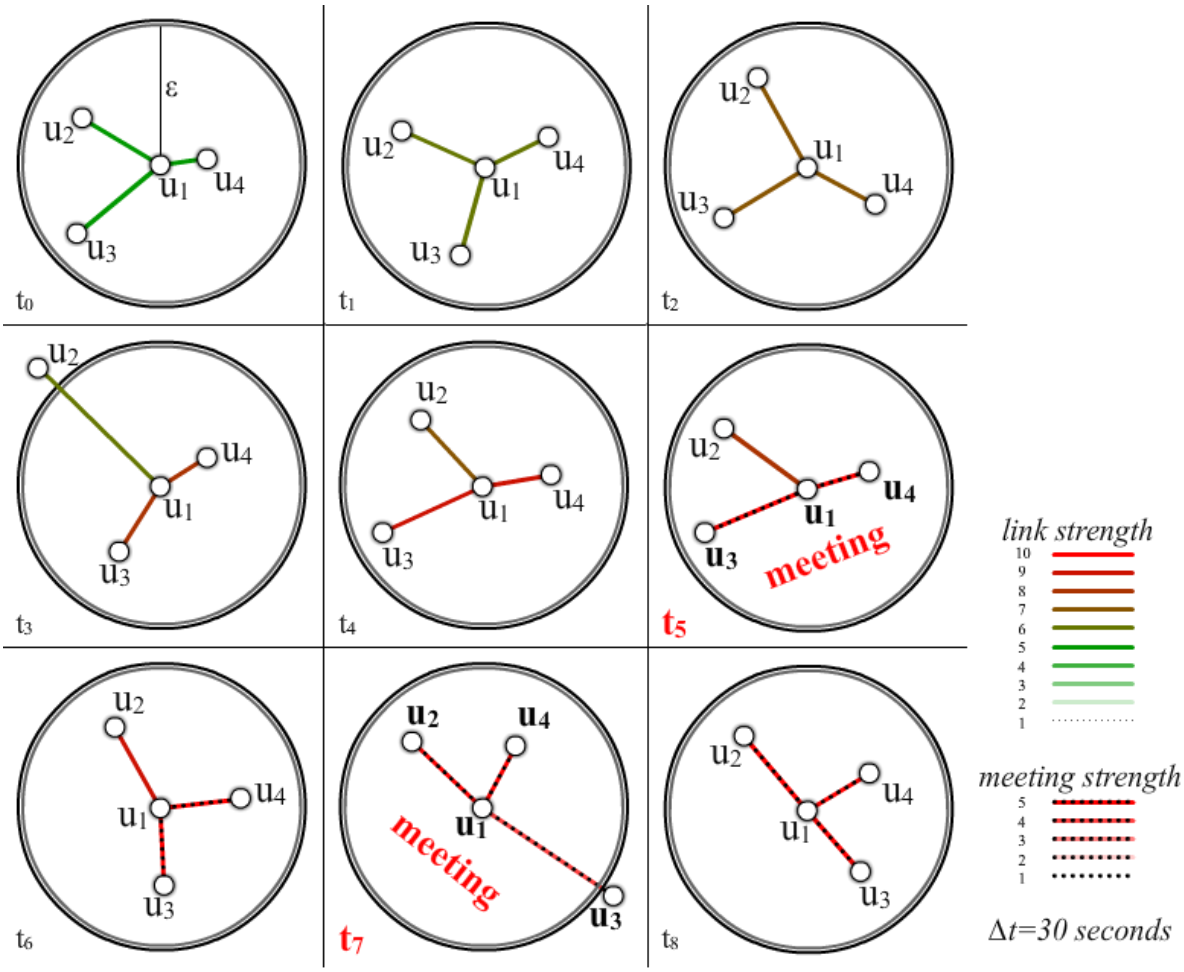


Figure 33: User-to-user meeting detection

In Figure 33 we notice how at t_0, u_1 is linked to the three other users: u_2, u_3 and u_4. The strength values increase at each time step for links where users are near each other. At t_3 we notice the link between u_1 and u_2 is slightly weakened and as a consequence, at t_5, u_1 is not part of the group meeting between u_1, u_3 and u_4. After one minute (at t_7) all four users are meeting.

Although user u appears to be *tying* the others together as part of a group, it is not really the case. The example omits displaying links which do not concern u_1 for the sake of simplification. If the traveling users create a pattern, such as people cycling on a narrow street or traveling by train, this solution alone will not work. Because of this, we prefer to have the nearby users defined in another way. Figure 34 shows users u_1 and u_7 linked because intermediate links exist between them. It is natural that the seven users in the image should be part of the same group even though not all users are inside a ε threshold (25 meters in MOPSI). To find these groups and update the user links we use agglomerative clustering. We describe this process in the next subsections.

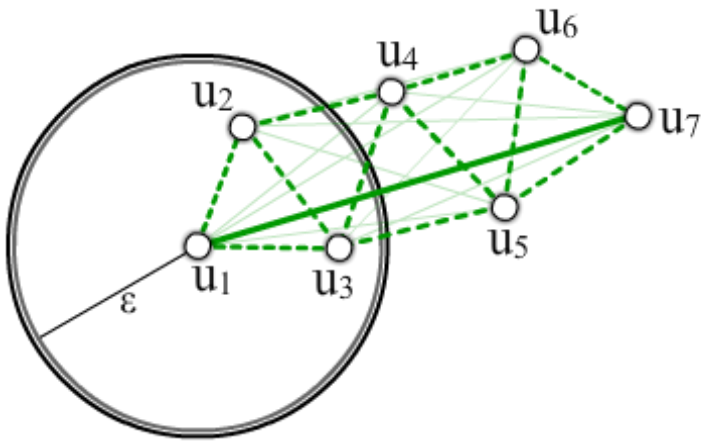


Figure 34: Nearby users

3.7.1 Clustering overview

Clustering is a method of unsupervised learning that given a data set of *D*-dimensional *N* data points $\boldsymbol{X} = \{\boldsymbol{x}_i\}\big|_{i=1}^{N},\ \boldsymbol{x} \in \mathbb{R}^{\mathrm{D}}$, partitions the data set into *M* clusters (groups) based on similarity measures, so that objects inside a cluster are similar to each other while dissimilar to points in the other clusters. Figure 35 shows the result of a cluster analysis where three clusters are detected. Partition $P = \{p_j\}\big|_{j=1}^{N},\ p_j \in \{1, 2, \dots, M\}$ defines the clustering by giving for each data point the index of the cluster where it is assigned to. A cluster S_a is defined as the set of data points that belong to the same partition:

$$S_a = \{\boldsymbol{x}_i | p_i = a\}.$$

The clustering is represented by the set $S = \{S_i\}\big|_{i=1}^{M}$, where *M* is the number of clusters.

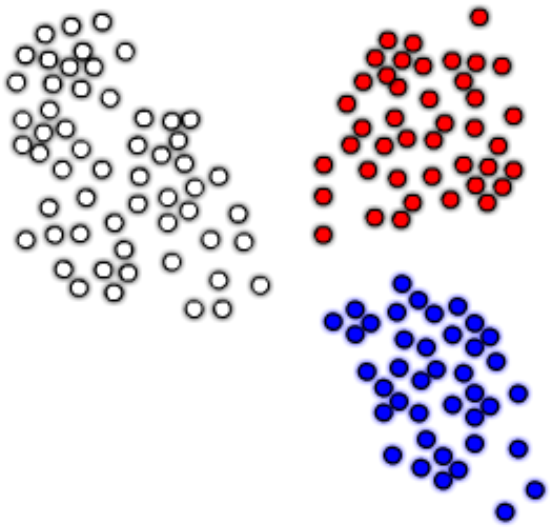


Figure 35: Clustering with |*S*|=3

Hierarchical clustering seeks to build a hierarchy of clusters. There are two approaches of doing this:

- Agglomerative clustering;
- Divisive clustering.

Agglomerative clustering [20] is a bottom up approach where initially all the data points are clustered individually $p_j = j,\ j = 1, \dots, N$. Then an iterative process starts where at each step, two clusters are selected based on a distance criterion to be merged. *Divisive clustering* [21] is the top down approach, starting with everything in one cluster, $p_j = 1,$

$j = 1, \ldots, N$, and then performing a split at each step. Both methods yield a hierarchy as illustrated in Figure 36.

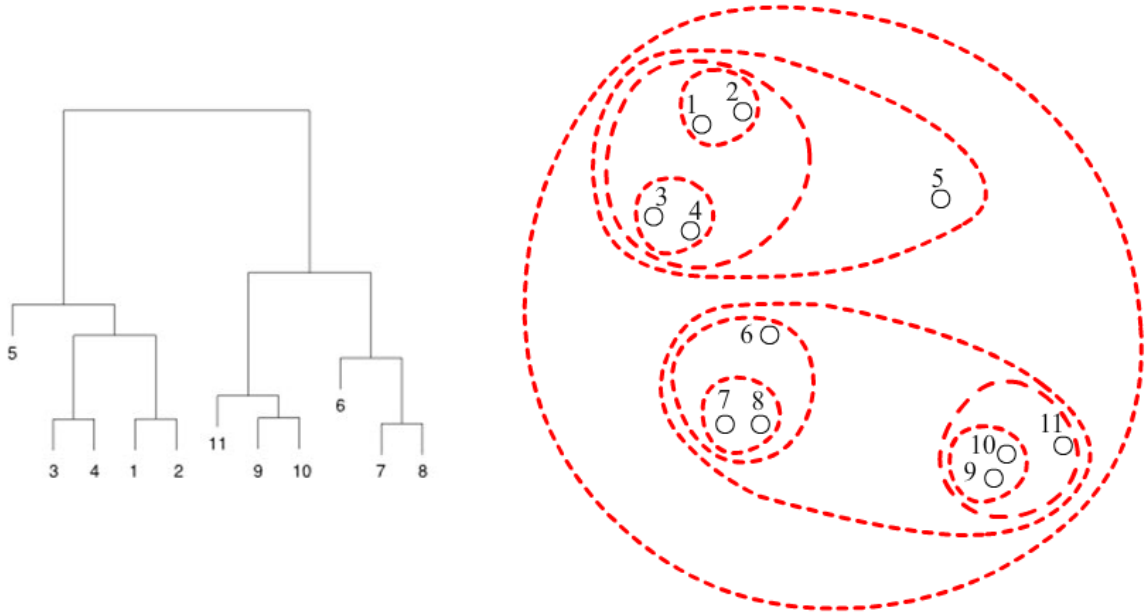


Figure 36: Hierarchy of clusters are shown using a dendrogram (left) and nested clusters (right) for agglomerative and divisive clusters accordingly

3.7.2 Single-Link Clustering

Single-link method is an example of agglomerative hierarchical clustering. Initially, each point is placed in a separate cluster, and at each step two clusters are selected for merge. The selection of the two clusters is made based on proximity. For the single link method, the distance of two clusters is defined as the minimum of the distance between any two points in the two clusters:

$$D(S_a, S_b) = \min_{\boldsymbol{x} \in S_a, \boldsymbol{y} \in S_b} d(\boldsymbol{x}, \boldsymbol{y})$$

Here S_a and S_b are two clusters and d is a merge criterion, which will be discussed in the next subsection. A $N \times N$ distance matrix is used to avoid recalculating the distances at each step. Consequently the matrix needs to be updated when two clusters merge by deleting the rows and columns corresponding to S_a and S_b and adding a new row and column for the newly formed cluster S_{ab}. The algorithm assumes we know the number of clusters M. A pseudocode for the single-link method can be written as follows.

```
SingleLink(X, M) -> S
Step 1. Calculate distance matrix for all data points.
Step 2. REPEAT
     Step 2.1: Find closest pair (Sa,Sb) to be merged.
     Step 2.2: Merge the pair (Sa,Sb) -> Sab.
     Step 2.3: Update distance matrix.
UNTIL |S| = M
```

Figure 37 illustrates the steps of the single-link method. In step 1, a cluster is created for each data point. Step 2 shows merging of two clusters. The algorithm repeats the process of merging two nearest clusters until all the points are inside a single cluster (at step 8). This builds the entire hierarchy. Typically the entire hierarchy is not needed and the algorithm simply stops when $|S| = M$.

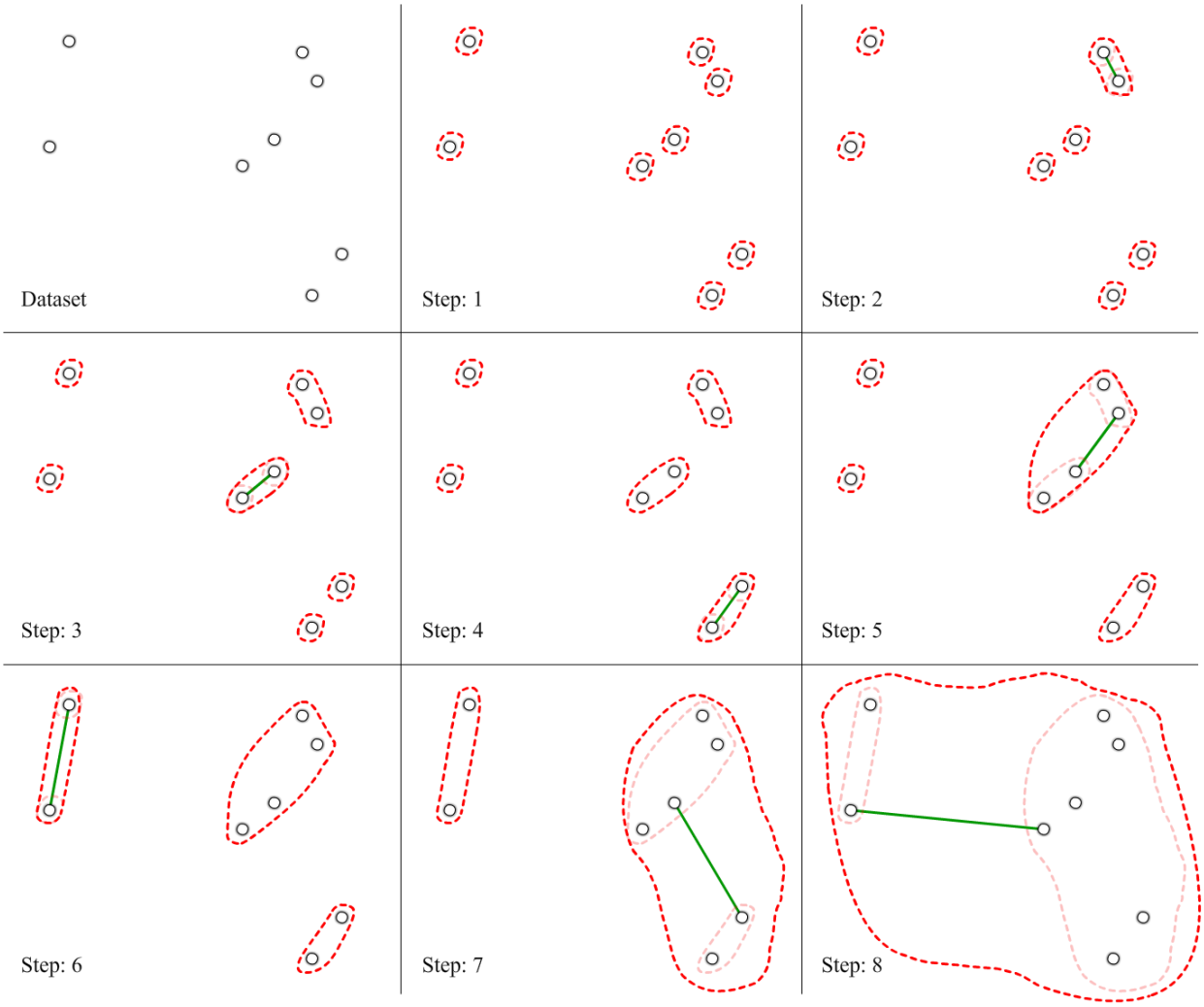


Figure 37: Single-Link clustering

In order to detect nearby users in MOPSI, we use the single-link method. We do not know the number of clusters. Instead, the algorithm stops when the distance between the closest cluster pair exceeds a threshold ε=25 meters. In this way, every point in a cluster is connected to at least one other point of the same cluster by a link of at most ε meters. This corresponds to our definition of the groups formed by nearby users illustrated in Figure 34.

Pseudocode of the single link method is below:

```
SingleLink(X, ε) -> S
Step 1. Calculate distance matrix for all data points.
Step 2. REPEAT
     Step 2.1: Find closest pair (Sa,Sb) to be merged.
     Step 2.2: d <- Distance between the closest pair.
     Step 2.3: IF d > ε THEN
          Step 2.3.1: Merge the pair (Sa,Sb) -> Sab.
          Step 2.3.2: Update distance matrix.
     Step 2.4: END IF
UNTIL d > ε
```

Figure 38 shows a dataset clustered using the single-link method, stopping under the ε threshold. By definition, a single point cannot form a group. Each remaining cluster is a group of users. For every pair of users in every group links will be updated. The exact method is shown in subsection 3.7.4.

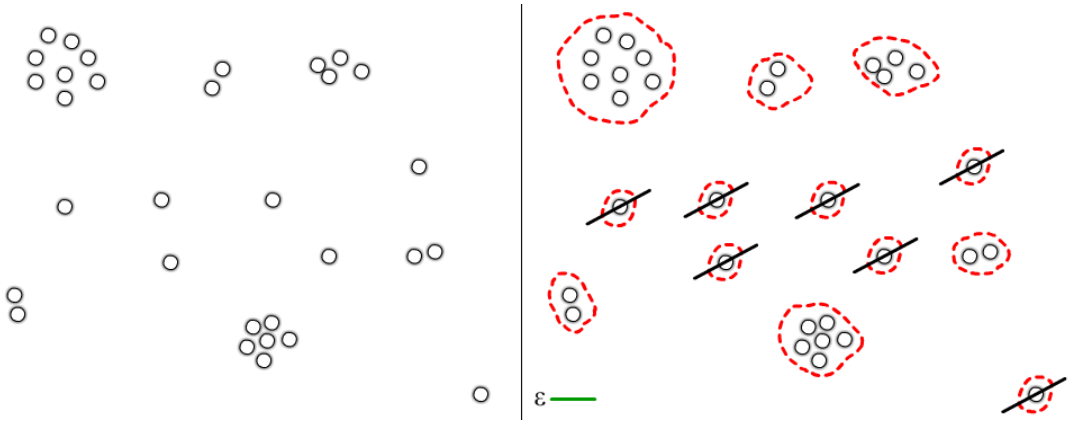


Figure 38: User groups

Single-link as described above can be easily implemented with time complexity of $O(N^3)$. Algorithms with better time complexity exist for mean square error criterion. In [22], the authors use a *k-nearest neighbor graph* to reduce the number of distance calculations. Time complexity of their proposed algorithm is $O(\tau\ Nlog\ N)$ where τ is the

number of nearest neighbor updates per iteration. In MOPSI, N corresponds to the number of online users.

3.7.3 Distance function

Distance calculation needs to take into account mobile users. Point to point distance calculated using Formula 7 will not work as expected. Let us consider two users moving together at a constant speed, sending location updates to the server every 30 seconds. Usually the users do not send this data at the same time as that would imply that they start the application at the same time. Let us assume a 10 second offset.

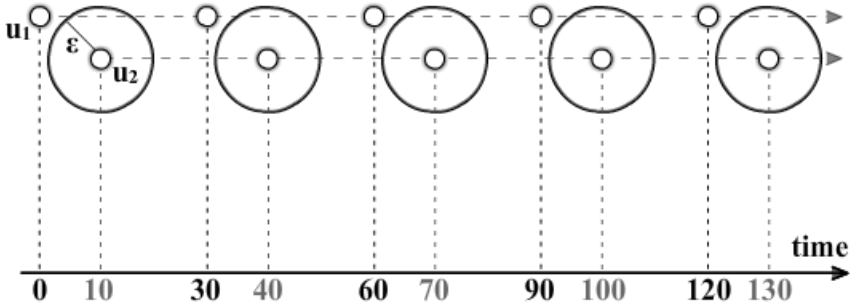


Figure 39: Users moving together with 10 second offset in location updates

In Figure 39 we observe that because of the offset, the users never appear to be less than ε meters away (25 meters in MOPSI), even though traveling together. Simply increasing the value of ε does not offer a good solution. Depending on transportation means, in 10 seconds a user can travel a significant distance (500 meters by train and even more than a kilometer by plane). Increasing ε so much will yield many miss-classifications.

As shown in Figure 40, we use interpolation in order to approximate the location of u_1 at the moment when we know the location of u_2. In other words we need to find $M(x, y)$, when knowing the location and time at *A*, *B* and *C*. To achieve this, the following formulas are used

$$k = \frac{t_A - t_C}{t_B - t_C}, \qquad x = \frac{x_A - kx_B}{1 - k}, \qquad y = \frac{y_A - ky_B}{1 - k}$$

Where

- t_P is the times component of point *P*;
- x_P and y_P are the location components of point *P*.

In our example $k = -\frac{1}{2}$ because of the 10 second offset between points *A* and *C* and the 30 second update interval between points *A* and *B*.

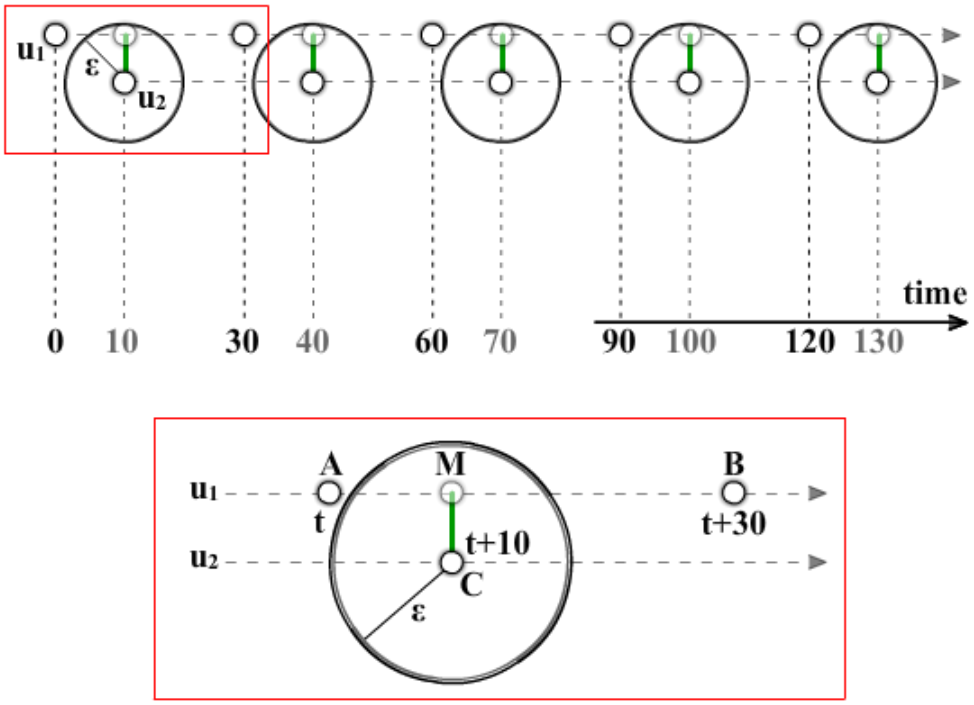


Figure 40: Interpolation method

3.7.4 Updating links

Based on the groups of users resulting from clustering, we can increase the strength of the links at each time step. A linked list is assigned to every user to keep track of the user-to-user links and the associated strength values. Each node of the list represents a user-to-user link and contains two pieces of information: a pointer to the linked user and a strength value.

In Figure 41 we consider four users at t_0. None of the users are inside a cluster and because of this the linked lists assigned to each of the users are empty. At t_1, u_2 and u_3 are clustered together and as a result the linked list assigned to u_2 has one element pointing to u_3 and showing a strength value of five. At t_2, u_1 and u_4 are also clustered together and as a result u_4 appears in the linked list assigned to u_1. Meanwhile, the $u_1 – u_3$ link strength increases to six. At t_3, the $u_1 – u_2$ link is created and added to the linked list of u_1. At this time the other existing links are strengthened.

The link is reversible, meaning that if u_1 is linked to u_2 then u_2 is linked to u_1. Because of this we store the information just once and assign it as a node to the linked list of the user with smaller user index in database. This is why at t_3 the linked list assigned to u_4 is empty even though u_4 is linked to u_1. In fact, because u_4 has the highest user index in the example, the assigned linked list will always be empty.

Linked lists are used instead of an adjacency matrix in order to save memory space. Because of the reversible property of the link, half of the matrix would not be necessary since it would only store redundant information. In addition, the user-to-user links are few and would result in having a sparse matrix.

At t_6 a user-to-user meeting is established between u_2 and u_3 because the link strength reaches the maximum value of 10. At t_7 the $u_1 – u_4$ link is upgraded into a meeting and at t_8 a user – user meeting is formed between u_1 and u_2.

Storing the user-to-user meetings is possible, however, it would require quadratic memory, and further processing when the group of users is later required. Instead we store the group of users meeting at a certain time. For example, the group $\{u_1, u_2, u_3, u_4\}$ is stored at t_7.

The method for detecting the meeting groups starts by defining a graph where edges are the user-to-user meetings. The task then becomes equivalent to the problem of finding connected components. A detailed description follows in the next subsection.

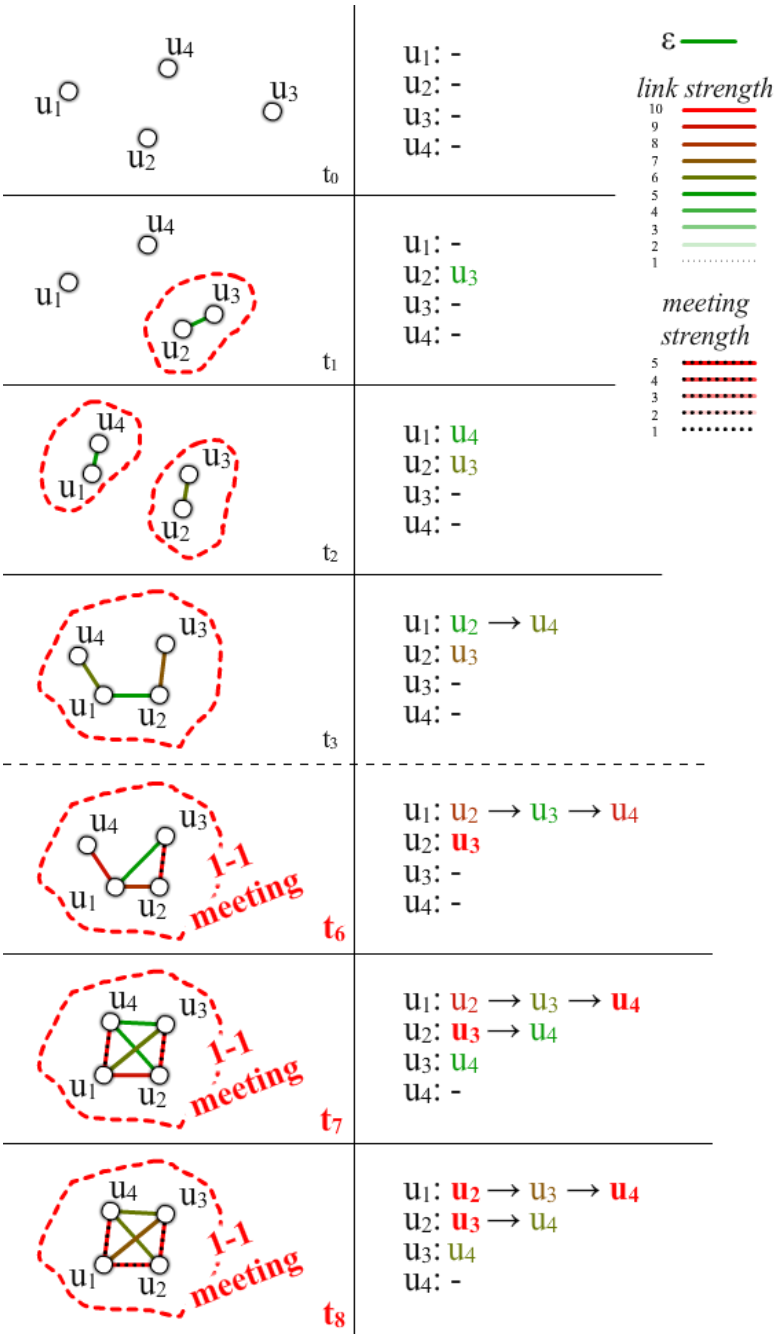


Figure 41: Increasing link strength

In Figure 42, the users: u_1, u_2 and u_3 are clustered together at t_0 and a link with strength 5 exists between u_1 and u_4. At t_1, u_1 and u_4 are not clustered together so the $u_1 - u_4$ link has to weaken. We handle the link strength updates in two steps:

- Increase the strength of the links between the users in the same cluster by 2;
- Decrease the strength of all links by 1.

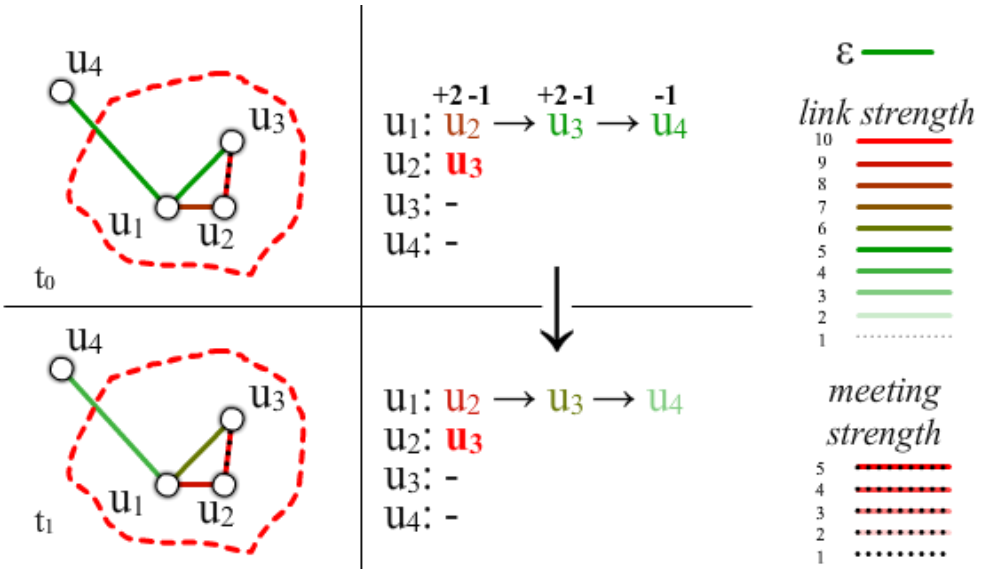


Figure 42: Strength increase for u_1-u_2, u_1-u_3 and decrease for u_1-u_4

The effect is that links between nearby users are strengthened by 1 and all other links are weakened by 1. The maximum possible number of user-user links is $\frac{n(n-1)}{2}$, when all users in the system are clustered together. Therefore, updating the strength value of the links has a time complexity of $O(n^2)$.

3.7.5 Detecting meeting groups

A subsample of the linked lists which contains just the user-to-user meetings are used to form adjacency list and define a graph (Figure 43). The edges of the graph are the one to one meetings. The task of finding the groups of users that are meeting is equivalent to searching for the connected components of the graph

It is straightforward to compute the connected components of a graph in linear time using either breadth-first search[23] or depth-first search[24]. A search begins at some particular vertex in the graph and will find the entire connected component before returning. Marking the elements belonging already to a connected component is required. To find all the connected components, a loop through the vertices is made, starting a new breadth first or depth first search whenever the current vertex is not marked as part of any component. We choose to ignore connected components containing only one vertex. Therefore Figure 43 shows two connected components. The users meeting as part of a

[23] http://en.wikipedia.org/wiki/Breadth-first_search
[24] http://en.wikipedia.org/wiki/Depth-first_search

group are found inside each connected component which we use in order to identify two meetings:

Users: u_6 and u_7 are meeting.

Users: u_1, u_2, u_3 and u_4 are meeting.

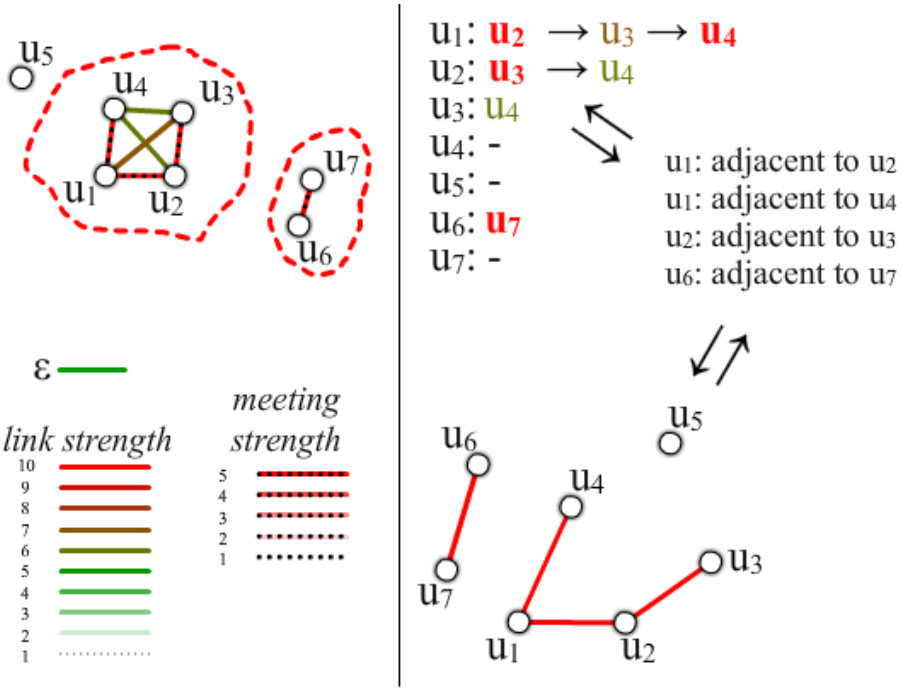


Figure 43: One to one meetings form a graph

3.8 O-MOPSI actions

O-MOPSI [23] is a mobile orienteering gaming system which provides a fun way to interact with the photo collection and services from MOPSI.

A game is defined by a set of goals that players need to reach as fast as possible and in no particular order. When all goals are visited, the game ends and time is used for ranking. In Figure 44 we see the results of the SciFest[25] O-MOPSI game of 2013. Anybody can play the game by downloading a mobile version of O-MOPSI from the mobile downloads page, http://cs.uef.fi/mopsi/mobile.php. If no games exist nearby, anybody can create own game using the O-MOPSI website.

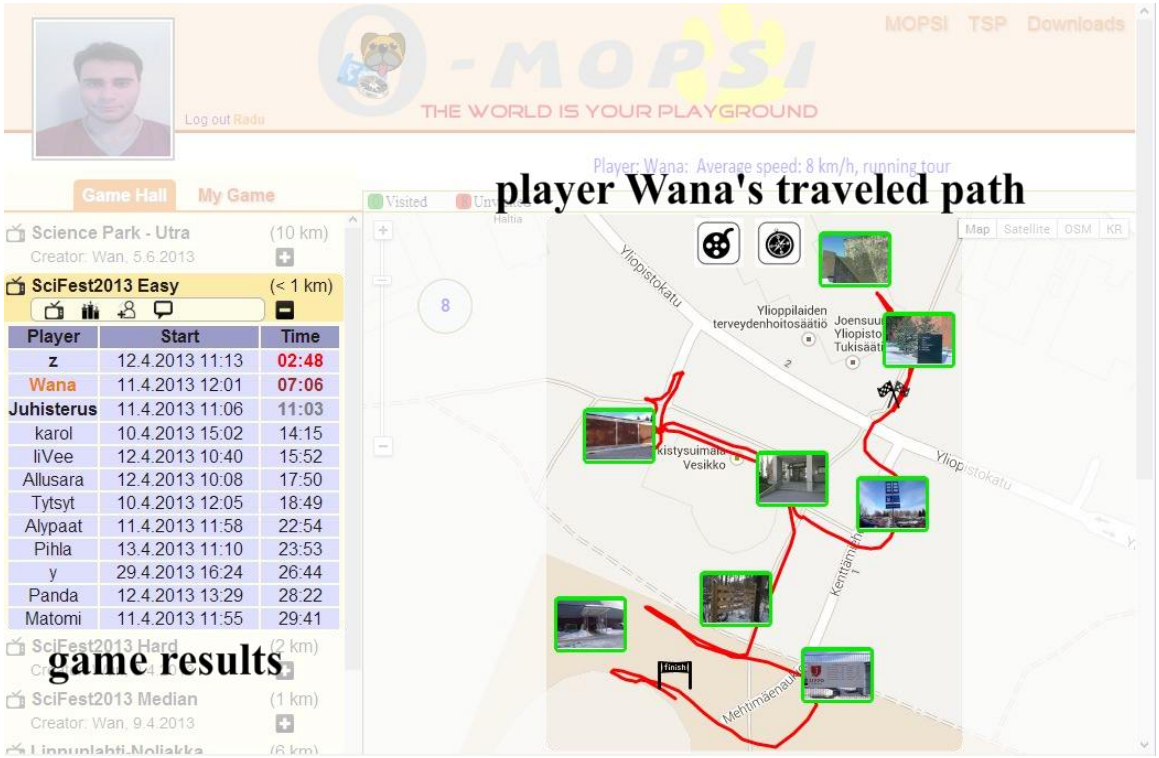


Figure 44: O-MOPSI website[26] showing results of the *SciFest2013 Easy* game

From the game actions, we consider the following four as relevant:

- Creating a game;
- Starting to play a game;

[25] http://www.scifest.fi/
[26] http://cs.uef.fi/o-mopsi/

- Finishing a game;
- Breaking a record;

The action of *creating a game* is *triggered* when a user creates a new O-MOPSI game by selecting a set of goals from the MOPSI photo collections and services. For example:

Radu created a game Severin 1.

O-MOPSI users can choose to play any created game. When a user joins a game the *started to play* action is *triggered*. For example:

Pasi started to play SciFest2013 Hard.

When the last goal is reached the game is considered finished and two actions may occur based on time. If a best time is achieved the *break record* user action is triggered, otherwise the *finish game* user action is triggered. Some examples are:

Noora broke the record for SciFest_Easiest.
Mikko finished Areena Long Track 2.

4 Notifications

Notifications are a subset of actions that are relevant to a user at a given time. They have three important motives. The first is to point out actions at the moment they are happening. This way, users know what others are doing in real-time. The second motive is to present an easy to access summary of what friends have been doing recently. Third motive is to help users to navigate inside the application by providing shortcuts to the respective action.

4.1 Push notifications

Push notifications are messages sent by the server in real-time. If all actions were included, the number of notifications can be overwhelming when many users are active. In addition, some user actions might not be of interest to other users. Because of these reasons we filter the actions so that only a subset reaches a particular user.

The push notification system is implemented based on the *publish/subscribe* architecture. A user can subscribe to an information channel provided by the server. Whenever new actions are available on one of the channels, the server pushes that information to the user. In MOPSI there is a channel for each action type:

- Taking/Uploading photo;
- Completing tracking;
- Changing city;
- Visiting service;
- Passing by service;
- Leaving service;
- Users meeting;
- O-MOPSI actions.

A user may subscribe to each channel individually. In addition to the filtering by action type, there is a possibility to filter by distance. A user may set a value in kilometers (x) so that if an action happens farther away than x km it will not be notified. For example, a user can choose to get notifications of users meeting only if they are less than 5 kilometers away so that he or she could participate in that meeting. Another example is the O-MOPSI *create game* user action so that a user is notified when new games are created nearby.

Major mobile operating systems (Android, iOS and Windows Phone) provide a service for handling push notifications. All platforms are similar in functionality. In the following we describe how the Windows Phone push notifications service[27] is used by MOPSI.

The Windows Phone push notification service is an *asynchronous*, *best effort* service that offers third-party developers a channel to send data to a Windows Phone app (MOPSI mobile application) from a cloud service (MOPSI server) in a power-efficient manner. It is an asynchronous service because when a new notification appears it does not block the interface and allows a user to continue doing what he/she was doing. It is a best effort service because the notifications are not delivered with 100% certainty. It is possible that some notifications never reach their destination.

Figure 45: Using the Windows Phone notifications framework

Figure 45 shows how a push notification is sent to the MOPSI mobile application. The process consists of the following steps:

1. MOPSI requests a push notification URI from the Push client service;
2. The Push client service negotiates with the Microsoft Push Notification Service (MPNS), which returns a notification URI to the Push client service. For example: http://db3.notify.live.net/throttledthirdparty/01.00/AAFW10y...MkUxREQ
3. The Push client returns the notification URI to the MOPSI application;
4. The MOPSI application sends the notification URI to the MOPSI server. Here the URI is associated to the MOPSI user;

 ⇦ http://db3.notify.live.net/throttledthirdparty/01.00/AAFW10y...MkUxREQ

[27] http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558(v=vs.105).aspx

5. When the MOPSI server needs to send information to a specific user that is not online it uses the associated URI to send a push notification to MPNS;
6. MPNS routes the push notification to the mobile device.

Depending on the format of the push notification and the payload attached to it, the info is delivered as raw data to an application, the app's Tile is visually updated, or a toast notification is displayed (Figure 46).

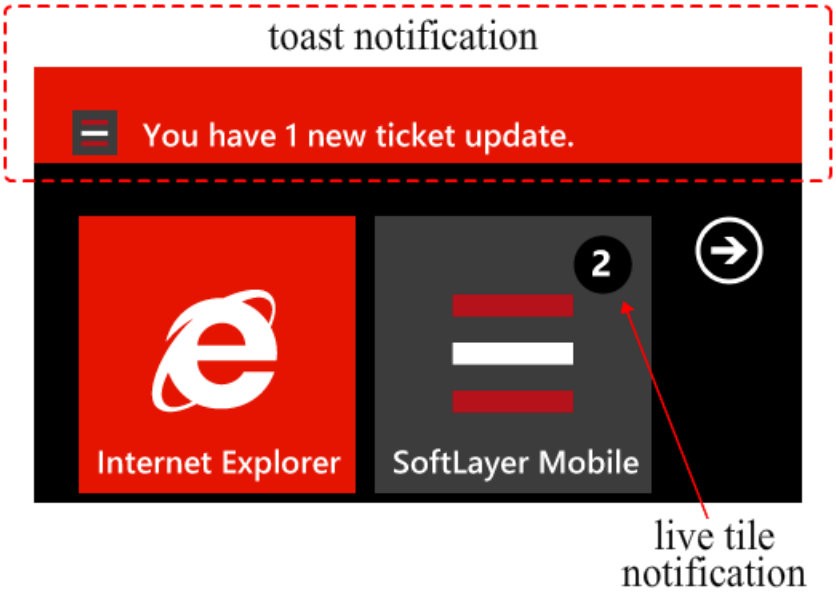


Figure 46: Toast and Tile notifications on SoftLayer[28] application

In MOPSI we use the toast notification to display the notification even when the application is closed. In Figure 47, on the left, a Windows Phone toast notification is shown. Tapping the notification opens the MOPSI application and in this case displays *Radu's* photo (right).

MPNS returns a response code to the MOPSI server after a push notification is sent indicating that the notification has been received and will be delivered to the device at the next possible opportunity (when device is on and internet connection exists). Although MPNS does not provide an end-to-end confirmation that the push notification was

[28] http://www.softlayer.com

delivered to the phone, it is possible for MPNS to return a response or error code to the MOPSI server which indicates that the device is unreachable and the notification will not be delivered yet.

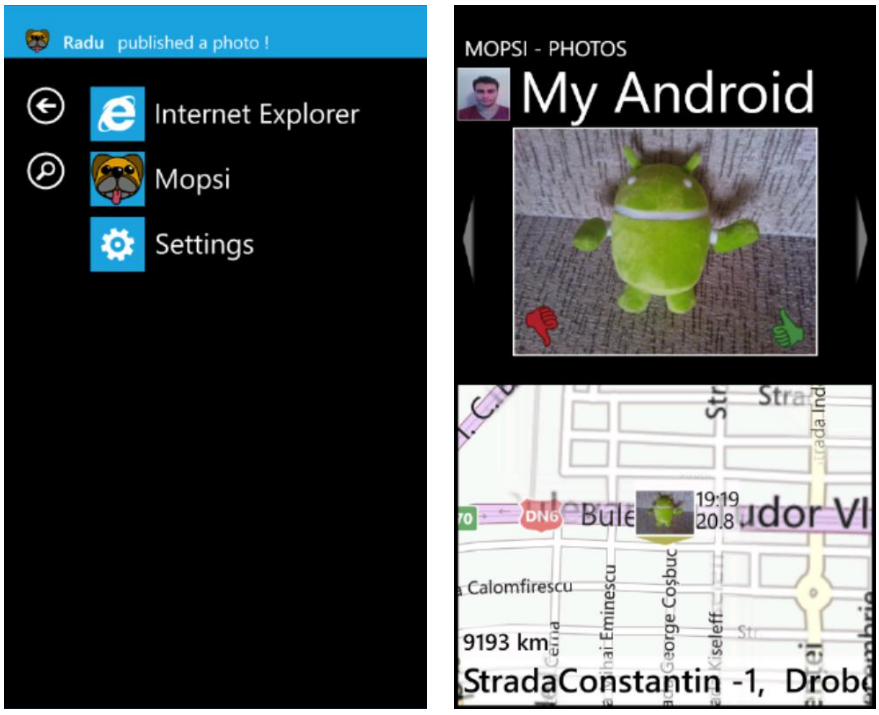


Figure 47: Toast notification (left) and MOPSI opened to show the content (right)

4.2 Pull notifications

Pull notifications are requested by the client side (website or mobile). This request can happen periodically or upon request. In the MOPSI mobile application, recent notifications are requested upon application start. Every 30 seconds a request is made to the server to get updates in notifications. In Figure 48 we see that when new notifications exist, the Windows Phone version of MOPSI shows the actions button with a red color. A sound is also played the moment new notifications are retrieved.

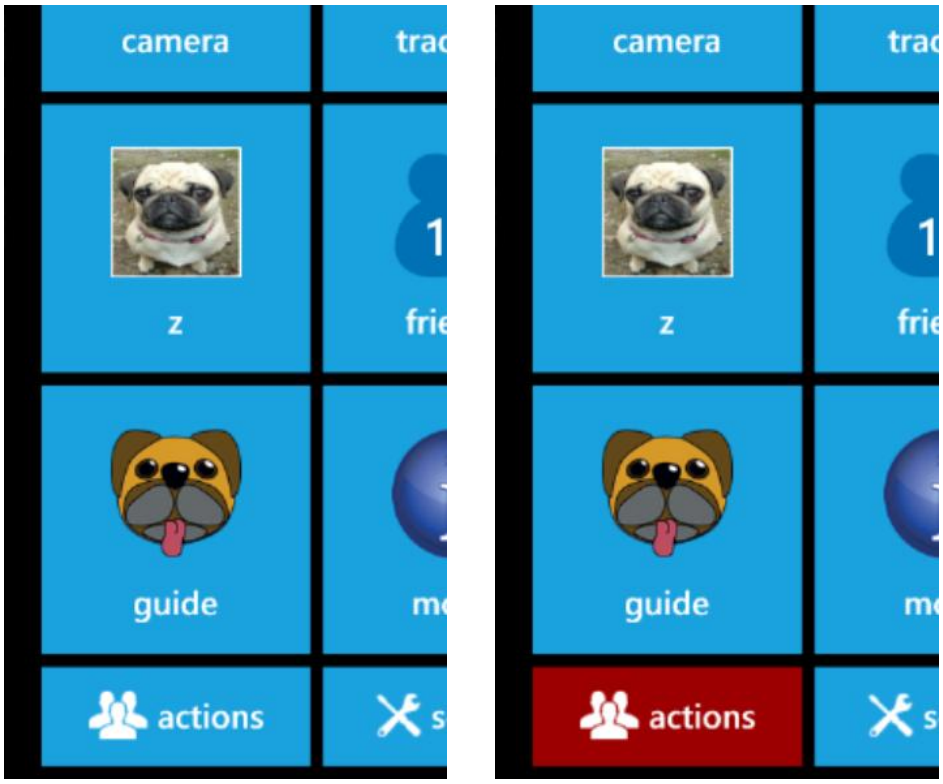


Figure 48: Red actions button when new notifications

Pressing the button opens the actions page (Figure 49), were the user can scroll through the list of recent actions. This is a summary of what friends were doing, sorted by time. Tapping any element of the list has a different behavior depending on the action type. For example, when tapping the *Radu took a photo* element, the photo is opened in the photo viewer. On the other hand, tapping *Pasi passed by Aura* shows the service details of the *Aura* restaurant. In addition to the notifications displayed inside the *actions* page, the *new* symbol is used on the *friends* page to indicate unseen elements from a particular user collection. Users *Oili* and *Pasi* have new elements in their collections (elements not seen yet by the mobile user).

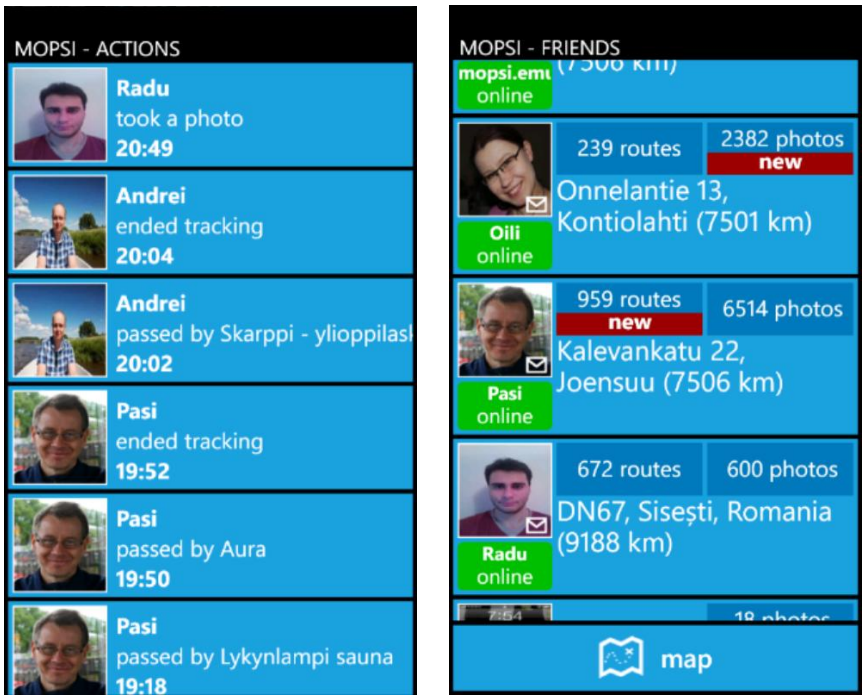


Figure 49: Recent user Actions (left) and new collection elements (right)

The actions page is useful because it provides navigation shortcuts inside the MOPSI application. To see the photo that *Radu* has taken requires only a single tap. Otherwise the photo should be opened using the following four steps (Figure 50):

1. Go to the friends page
2. Scroll to find *Radu* in the list
3. Open *Radu's* photos
4. Scroll in *Radu's* photo collection to find the photo if not the first in the list

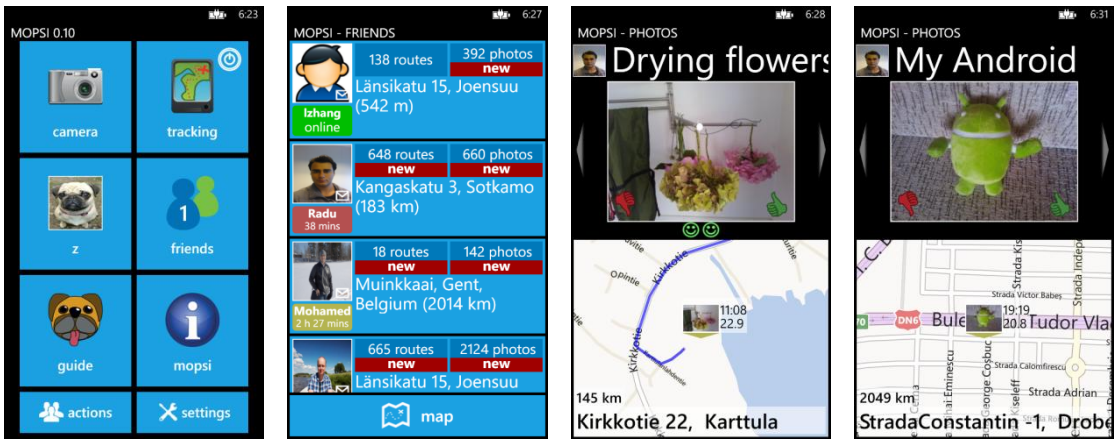


Figure 50: Steps to open a certain photo when not using the actions page

To find information about the *Aura* restaurant without using the actions page shortcut is also possible by following the following five steps:

1. Go to the guide page
2. Enter the keyword *Aura*
3. Tap the search button
4. Scroll to find the *Aura* service in the list
5. Tap the *Aura* service result

The elements in the actions page are not always important to the MOPSI users, however, when they are, the time needed to get the needed information decreases substantially.

On the website, the three most recent notifications are shown under the users tab (Figure 51). The hyperlinks are clickable and have the same effect as on the mobile devices. Clicking *Pasi*, *Jukka* or *Minttu* opens their respective profile pages, clicking *Valintatalo* shows the service information where *Radu* passed by, clicking *Completed Walking tour of 1 km 25 m* will show *Mohamed*'s walking route and clicking and so on. The list is refreshed every 30 seconds to display new notifications. A more extended view can be seen on user's profile pages where 20 notifications are shown.

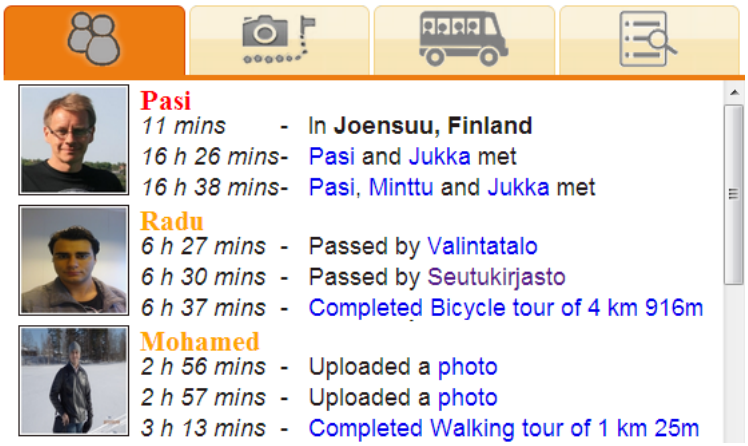


Figure 51: MOPSI website showing the three most recent actions for each user

Linking the MOPSI account to a Facebook[29] account is possible (Figure 52). Doing this, selected user actions can be shared with the friends outside MOPSI. For example, when new photos or routes are published to Facebook, the notification system from Facebook shares it with Facebook friends.

[29] http://www.facebook.com

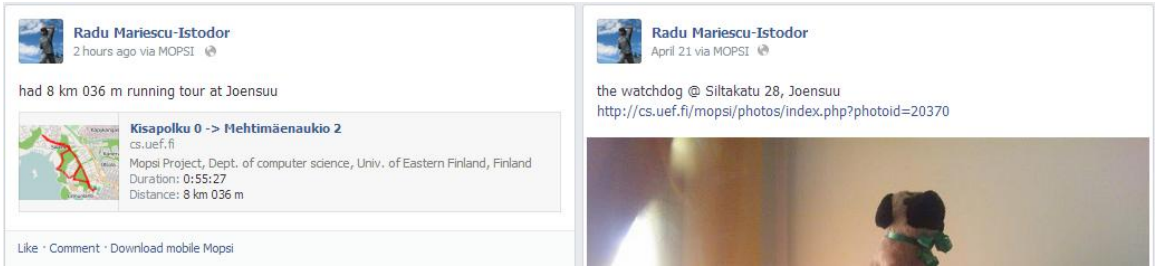
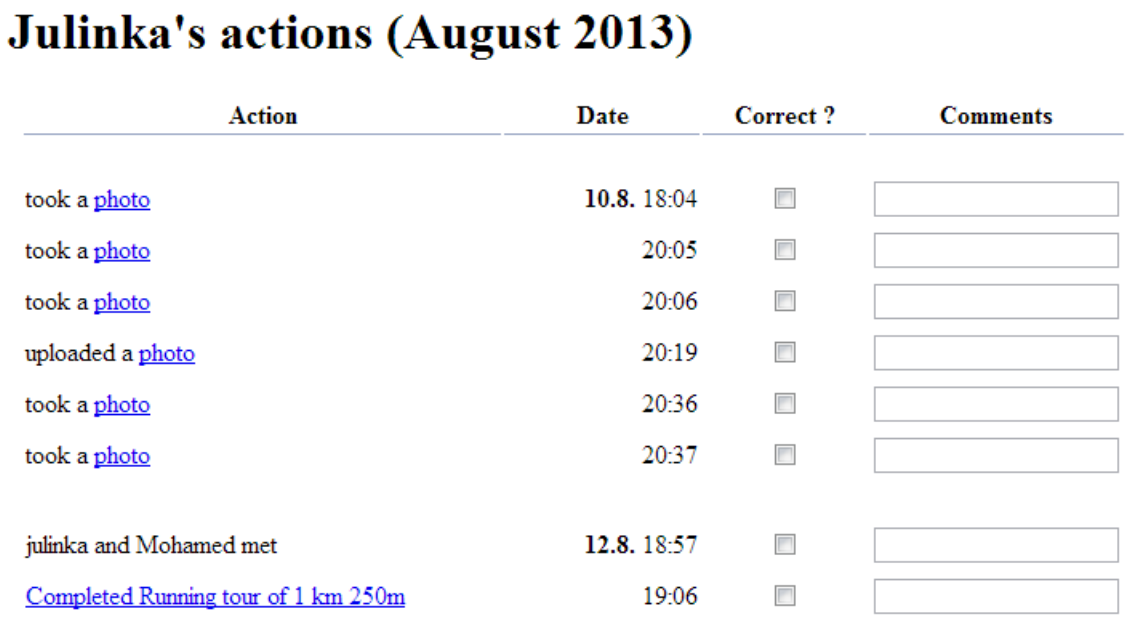


Figure 52: MOPSI account sharing information on Facebook

5 Experimental results

We evaluate the action detection system using real information collected from MOPSI users during the most recent month August 2013, consisting of approximately 800 users in total. Ground truth[30] was collected from the nine most active users willing to participate in a survey.

Each user was asked to mark whether his/her actions identified by MOPSI matched to what actually happened. The actions were listed as illustrated in Figure 53. The location of the actions varied significantly as the users traveled in six countries: Finland, Romania, Germany, Latvia, Lithuania and the United Kingdom.



Julinka's actions (August 2013)

Action	Date	Correct ?	Comments
took a photo	**10.8.** 18:04	☐	
took a photo	20:05	☐	
took a photo	20:06	☐	
uploaded a photo	20:19	☐	
took a photo	20:36	☐	
took a photo	20:37	☐	
julinka and Mohamed met	**12.8.** 18:57	☐	
Completed Running tour of 1 km 250m	19:06	☐	

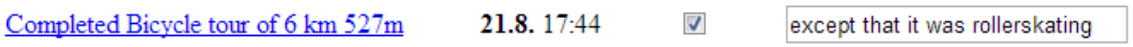
Figure 53: User *Julinka*'s actions from August 2013

If an action was correctly detected, users were asked to mark the checkbox. If the action did not happen or the information is inaccurate, users were asked to leave the checkbox unchecked and specify the reason in the *Comments* field. If the users do not remember of a particular action they were asked to leave the checkbox unchecked and leave the *Comments* field empty. In this way only confirmed information is used for the evaluation. For example in Figure 54, *Karol* marks the bicycle action true even if roller skating was the true action but it is not one of the recognized movement types detected in MOPSI and bicycle fits as the most similar mode of transportation. *Pasi* and *Minttu* were orienteering.

[30] http://cs.uef.fi/~radum/actionsGroundTruth/

They met at the start and finish, but did not meet in the forest during the orienteering despite MOPSI claims meeting detection. One reason for how this can happen is inaccurate GPS signal. Another reason may be too high threshold (25 meters) which combined with orienteering specific factors like visibility reduced by trees and the frequent checking of the map could mean that the two users did not see each other.

User *Karol*:



User *Pasi*:

Figure 54: Sample answers from the questionnaire

MOPSI recognized a total of 1197 actions during the one month interval (Table 2). The action count forms an uneven distribution. The most common actions detected were the photo taking or uploading. These are followed by the meeting, tracking, pass by, visit and leave actions in this particular order. The visit and leave actions are the fewest. This has mostly to do with the fact that MOPSI services currently only exist in Finland and the users were traveling outside the country when the actions for generating the ground truth were recorded. Another reason is that people usually visit services in weekend and spare time, while the other actions can appear more frequently.

Table 2: Number of detections for each action type per user

Action / User	Photo	Meeting	Tracking	Pass by	Visit	Leave	
Karol	7	5	5	10	0	0	27
Mikko	2	0	3	0	0	0	5
Mohamed	27	18	1	16	3	3	68
Oili	434	3	2	2	0	0	441
Mohammad	4	2	5	0	0	0	11
Pasi	247	30	27	13	2	2	321
Radu	17	3	2	0	0	0	22
Andrei	40	29	22	10	2	2	105
Julinka	125	14	30	26	1	1	197
	903	104	97	77	8	8	**1197**

We summarize results from the survey in Figure 55 where we calculate the detection error according to the action type. The error is calculated by counting the incorrectly detected user actions as a percentage from the total actions detected by the system. For example, 6.7% error for the meeting action is a result of 7 misdetections out of 104 detections in total.

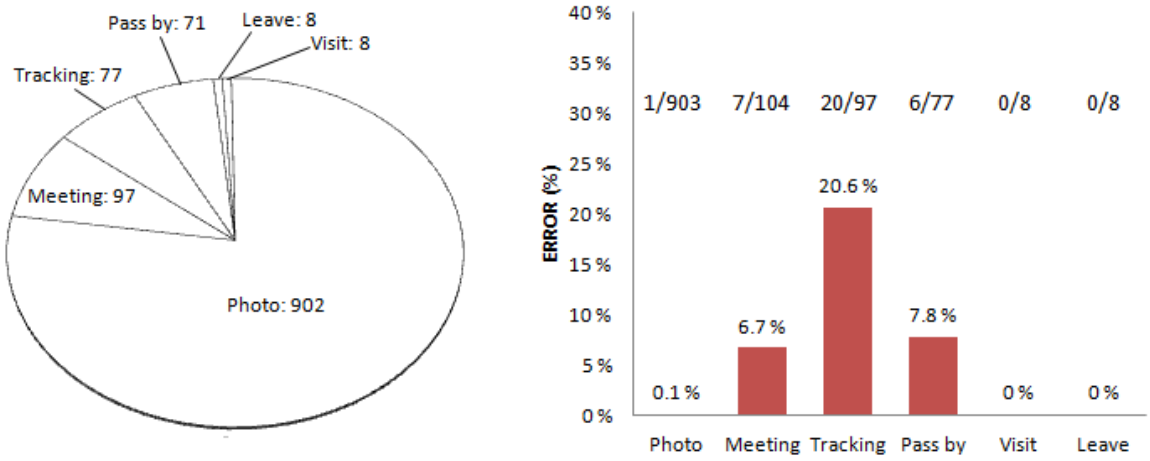


Figure 55: Distribution of correctly detected actions (left), and detection errors by action type (right)

We can calculate an un-weighted average error (*UA*) for the action detection. When a user performs an action at random, the probability of incorrect detection is estimated by the *UA* value. *UA* is obtained using the following formula:

$$UA = \frac{\sum_{i=1}^{N} e_i}{N}, \qquad e_i = \frac{c_i}{C_i}$$

where:

- N is the number of action types we perform the evaluation for,
- e_i is the error probability for the type i,
- c_i is the number of correctly detected actions of type i and
- C_i is the total number of detected actions of type i

Because of the uneven distribution of performed actions we also calculate a weighted average error (WA) using the formula:

$$WA = \frac{\sum_{i=1}^{N} c_i e_i}{\sum_{i=1}^{N} C_i}$$

We can see in Table 3 that *WA* is less than *UA*. This is because actions that happen most frequent present less error in detection. In MOPSI, photo actions represent 75% of all detected actions. The weight of these actions combined with a low error in detection (0.1%) help decrease the WA.

Table 3: Average error values

UA	5.9
WA	2.4

Figure 56 summarizes the detection errors for the nine users that participated in the survey. Detection error is below 20% for every user. The error for the top two most active users, *Pasi* and *Julinka* is also shown.

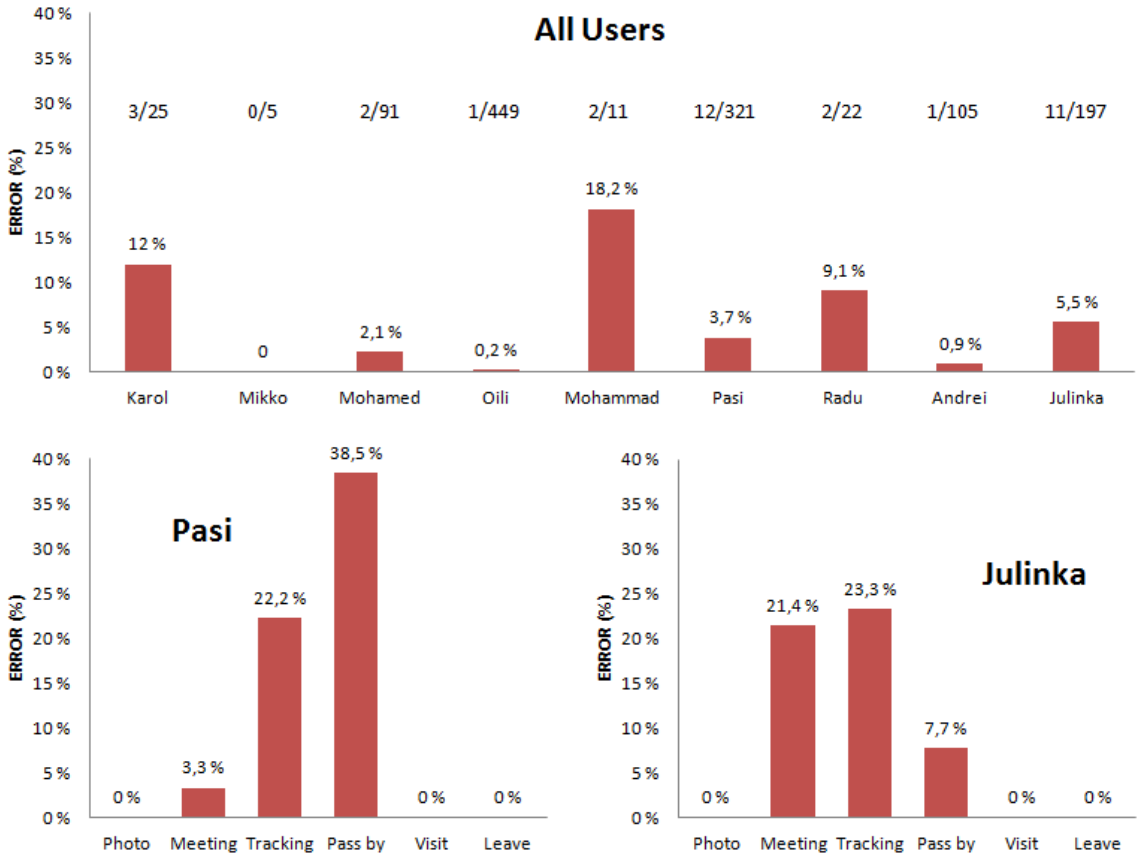


Figure 56: Error by user (top), user-specific errors by action type (bottom)

Taking or uploading photo are detected most accurately (error of 0.1%) because they are *triggered* actions, recorded when individual users send upload requests to the server. The requests contain the photo itself, description and the location. This is all that is needed to

recognize the action. If the photo information fails to reach the server, another attempt is made until the server confirms that the photo is in the database and the action is detected. Therefore, it is very rare that the action detection fails here. Situations in which failure can happen are: device running out of battery, device running out of memory and application crashes at a critical moment, which can lead to lost photos.

Meeting actions have a probability of 6.7% to be miss-detected (Figure 55). In Table 4 selected comments from users about three false meeting detections are shown. The comments identify two scenarios in which the detection fails. Firstly, users *Pasi* and *Karol* have the same problem: multiple meetings with the same person are detected instead of just one that lasted longer. The second situation happens when users are close to each other, but not meeting. This is identified by *Julinka* in her comment.

Finishing tracking is also a *triggered* action but the analysis of the movement type incorrectly is considered here as an error. Detecting them properly causes the most errors currently, relative to the other action types. Some typical errors are shown in Table 4.

The pass by is a *concluded* action for which variables such as GPS accuracy, service locations and area occupied by the services influence the correct detection. It is difficult to analyze why misdetections appear. In some cases, however, the reasons are known. For example, user *Karol*'s response on visiting a coffee place, the pass by happened because the previous day (when last time using GPS) *Karol* was around the coffee place. When the GPS sensor starts in a mobile device, it takes time before location is established, and the previous location is sometimes used until then. In this case, *Karol* appeared near the coffee place and our system interpreted this situation as a pass by. Similar situation happened for *Pasi* when he started the application on a device which he last used the previous day close to the restaurant.

Misdetection of the visit actions is less frequent. For a visit to be detected, the user needs to be close to the service for a considerable amount of time. This fact limits false detections such as the ones described by *Karol* and *Pasi* for pass by. We note, however, that only eight visits happened during the entire month, which is significantly smaller than that of other actions. The reason has partly to do with the fact that MOPSI services currently only exist in Finland and the users were traveling abroad when the actions for generating the ground truth were recorded. Another reason is that people usually visit services during weekend and spare time, while the other actions can appear more frequently. For example tracking can be done when going to and back from work. Here it is typical that meetings take place. Passing by happens implicitly when moving around the city and users usually take photos when something interesting, fun or noteworthy takes place.

The same situation applies for the leave action which implies that a visit needs to be detected first before leave can happen. More testing is therefore needed in the future.

Table 4: User comments on badly detected actions

User	**Meeting**	User comment
Pasi	Pasi and Minttu(19:15, 19:38, 19:46)	I think we were together from 19:15-19:46 entire time
Julinka	Radu and Julinka	probably, because we were neighbours
Karol	Andrei and Karol (19:21, 19:46, 20:17)	we never separated, were rollerskating all the time together
	Tracking	
Julinka	Completed Running tour of 1 km 250m	it was a bicycle tour
Pasi	Completed Car tour of 16 km 863m	32 km running tour!
Pasi	Completed Walking tour of 10 km 771m	Orienteering (Run+Walk)
Mohammad	Completed Running tour of 2 km 689m	I didn't do running, probably it has been cycling
	Pass by	
Pasi	Ravintola Martina	Time does not match. We were in Lehmo!!!
Karol	Kahvila Pilkku - Joensuun Seutukirjasto	not possible, because I wasn't in Joensuu

These results only measure the correctness of detected actions. It is very difficult to find what actions were missing and should have been detected. Users are not expected to know or remember what else they did during some time interval (all places they visited or passed by and the people they met for example). It might be possible for taking photos and tracking, however, no critical errors exist there.

6 Conclusions

In this thesis we described the *user action detection* and *notification* modules for the MOPSI[31] application. MOPSI is a locator assistant that helps individuals to know where their friends are and what is around them. It supports photo sharing, easy tracking, and chatting with friends. MOPSI application offers location-based services such as search, recommendation, data collection, public transport information, users tracking and the relatively new *user action notifications*. By *user action* we understand a user's behavior at a certain time. Location-based actions take the user location into account.

Part of the user actions are *triggered*, meaning that they are recognized upon a direct server request made by a single user. We can detect login, logout, taking or uploading photos, completing tracking and creating a service in this category. Other actions need to be *concluded* based on additional system information. These include changing city, visiting or passing by places and meetings between users. Recognizing the *concluded* actions is not trivial due to inaccuracy in GPS signal and user movement. We presented the *link method* as a way to effectively deal with inaccurate GPS data and avoid misdetection. We employed hierarchical clustering in order to detect groups of stationary or traveling users. To handle moving targets we use interpolation to predict a user's location at a time convenient to perform distance measurements.

A subset of these actions is relevant to some users at a given time. The job of the *notification* module is to deliver these actions to users which are interested in them. The motive of notifications is to point out actions at the moment they are happening. This way, users know what their friends are up to in real-time. Another motive is to present an easy to access summary of what friends have been doing recently. Third motive is to help users to navigate inside the application by providing shortcuts to the respective action.

To study the effectiveness of the action detection methods we created a ground truth database. The most active MOPSI users participated in a survey where they are asked to mark if actions identified by the system have indeed happened. The actions from the most recent month (August 2013) were listed for each user. The location of the actions varied significantly as during August MOPSI users traveled in the following countries: Finland, Germany, Latvia, Lithuania, Romania and the United Kingdom. The qualitative study showed that the system performs with an average action identification error of below 6%.

The study in this thesis opens up avenues in location based research. One of the directions is to find *location clusters*. This way we can acknowledge relevant places that do not exist explicitly in our database. Different location data can be used in the

[31] http://cs.uef.fi/mopsi

clustering. For example, if end points from the routes of a user are used, the clusters will point to frequent destinations such as home, workplace, etc. If photo locations are used instead, it is likely the clusters represent places worth to visit like parks, monuments, etc. When locations where users meet are clustered it is likely that the result will point to places where people socialize like bars, restaurants, etc.

Another possibility is to find *similar users* by analyzing their actions. User similarity aims to group users with common interests together in the hope of improving their social interaction. This would also improve the recommendation system described in [15]. For example if user A visits coffee shop C_1 and user B visits coffee shops C_2 and C_3 then users A and B are similar in the aspect of liking coffee shops. The recommendation system can use this data and recommend C_2 and C_3 to user A and C_1 to user B.

Detecting the *visit*, *pass by* or *leave* actions can be improved in several ways. Firstly, representing the services as polygons in the database can reduce false detections. The second would be to associate an estimated amount of time the user needs to use a given service will improve the classification. For example, the time spent at a drive-through can be a matter of minutes whereas visiting a restaurant often takes an hour or more. Currently this scenario will appear to be a pass by instead of a visit.

Another extension could be to change the *notification filtering* from manual settings to automatic based on the actions a user shows interest in. The general idea is to record the actions for which a given user did not present interest in the past and automatically filter similar actions out for that user in the future.

REFERENCES

[1] R. Helm, R. Johnson, J. Vlissides E. Gamma, "Observer," in *Design Patterns: Elements of Reusable Object-Oriented Software*.: Addison-Wesley, 1994, pp. 326-337.

[2] F.C. Gartner, O. Kasten, A. Zeidler L. Fiege, "Supporting mobility in content-based publish/subscribe middleware," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, New York, 2003, pp. 103-122.

[3] V. Cahill R. Meier, "On Event-Based Middleware for Location-Aware Mobile Applications," *IEEE Transactions, Software Engineering*, vol. 36, no. 3, pp. 409-430, 2010.

[4] N. Davies, K. Mitchell, A. Friday K. Cheverst, "Experiences of developing and deploying a context-aware tourist guide: the GUIDE project," in *MobiCom '00 Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, 2000, pp. 20-31.

[5] Y. Zheng, Q. Luo, X. Xie X. Xiao, "Inferring social ties between users with human location history," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-17, 2012.

[6] E.D. Nitto, A. Fuggetta G. Cugola, "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS," *IEEE Transactions, Software Engineering*, vol. 27, no. 9, pp. 827-850, 2001.

[7] M. Hauswirth, M. Jazayeri I. Podnar, "Mobile push: delivering content to mobile users," in *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, Vienna, 2002, pp. 563-568.

[8] D. Wunsch X. Rui, "Survey of clustering algorithms," *Neural Networks, IEEE Transactions*, vol. 16, no. 3, pp. 645-678, 2005.

[9] A. Tabarcea, M. Chen, P. Fränti K. Waga, "Detecting movement type by route segmentation and classification," in *Collaborative Computing*, Pittsburgh, 2012, pp. 508-513.

[10] A. Tabarcea, R. Mariescu-Istodor, P. Fränti K. Waga, "Real Time Access to Multiple GPS Tracks," in *International Conference on Web Information Systems &*

Technologies (WEBIST'13), Aachen, 2013.

[11] J. Kuittinen, A. Tabarcea, L. Sakala P. Fränti, "MOPSI location-based search engine: concept, architecture and prototype," in *ACM Symposium on Applied Computing (SAC'10)*, Sierre, 2010, pp. 872-873.

[12] P. Fränti, S. Mehta G. Hariharan, "Data Mining for Personal Navigation," in *SPIE Conference on Data Mining and Knowledge Discovery: Theory, Tools, and Technology IV*, Orlando, 2002, pp. 355-365.

[13] A. Tabarcea, J. Kuittinen, V. Hautamäki P. Fränti, "Location-based search engine for multimedia phones," in *IEEE International Conference on Multimedia & Expo (ICME'10)*, Singapore, 2010 , pp. 558-563.

[14] V. Hautamäki, P. Fränti A. Tabarcea, "Ad-hoc georeferencing of web-pages using street-name prefix trees," in *International Conference on Web Information Systems & Technologies (WEBIST'10)*, Valencia, 2010, pp. 237-244.

[15] J. Chen, A. Tabarcea P. Fränti, "Four aspects of relevance in sharing location-based media: content, time, location and network," in *International Conference on Web Information Systems and Technologies*, Noordwijkerhout, 2011.

[16] A. Tabarcea, P. Fränti K. Waga, "Recommendation of Points of Interest from User Generated Data Collection," in *Collaborative Computing*, Pittsburgh, 2012, pp. 550-555.

[17] A. Rosi, M. Mamei, F. Zambonelli L. Ferrari, "Extracting urban patterns from location-based social networks," in *3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, New York, 2011, pp. 9-16.

[18] R. Arkins, B. Segall P. Sutton, "Supporting Disconnectedness – Transparent Information Delivery for Mobile and Invisible Computing," in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, 2001.

[19] P. Fränti A. Kolesnikov, "Polygonal approximation of closed discrete curves," *Pattern Recognition*, vol. 40, no. 4, pp. 1282-1293, 2007.

[20] J.H. Ward, "Hierarchical Grouping to Optimize an Objective Function," *Journal of the American Statistical Association*, vol. 58, pp. 236-244, 1963.

[21] T. Kaukoranta and O. Nevalainen P. Fränti, "On the splitting method for vector

quantization codebook generation," *Optical Engineering*, vol. 36, no. 11, pp. 3043-3051, 1996.

[22] O. Virmajoki, V. Hautamaki P. Franti, "Fast Agglomerative Clustering Using a k-Nearest Neighbor Graph," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1875-1881, 2006.

[23] Z. Wan, K. Waga, P. Fränti A. Tabarcea, "O-Mopsi: Mobile Orienteering Game using geotagged photos," in *International Conference on Web Information Systems & Technologies (WEBIST'13)*, Aachen, 2013.

[24] H. Garcia-Molina Y. Huang, "Publish/subscribe in a mobile environment," *Wireless Networks - Special issue: Pervasive computing and communications*, vol. 10, no. 6, pp. 643-652, 2004.

[25] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal (British Computer Society)*, vol. 16, no. 1, pp. 30-34, 1973.

[26] W.H. Equitz, "A New Vector Quantization Clustering Algorithm," *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 37, no. 10, pp. 1568-1575, 1989.